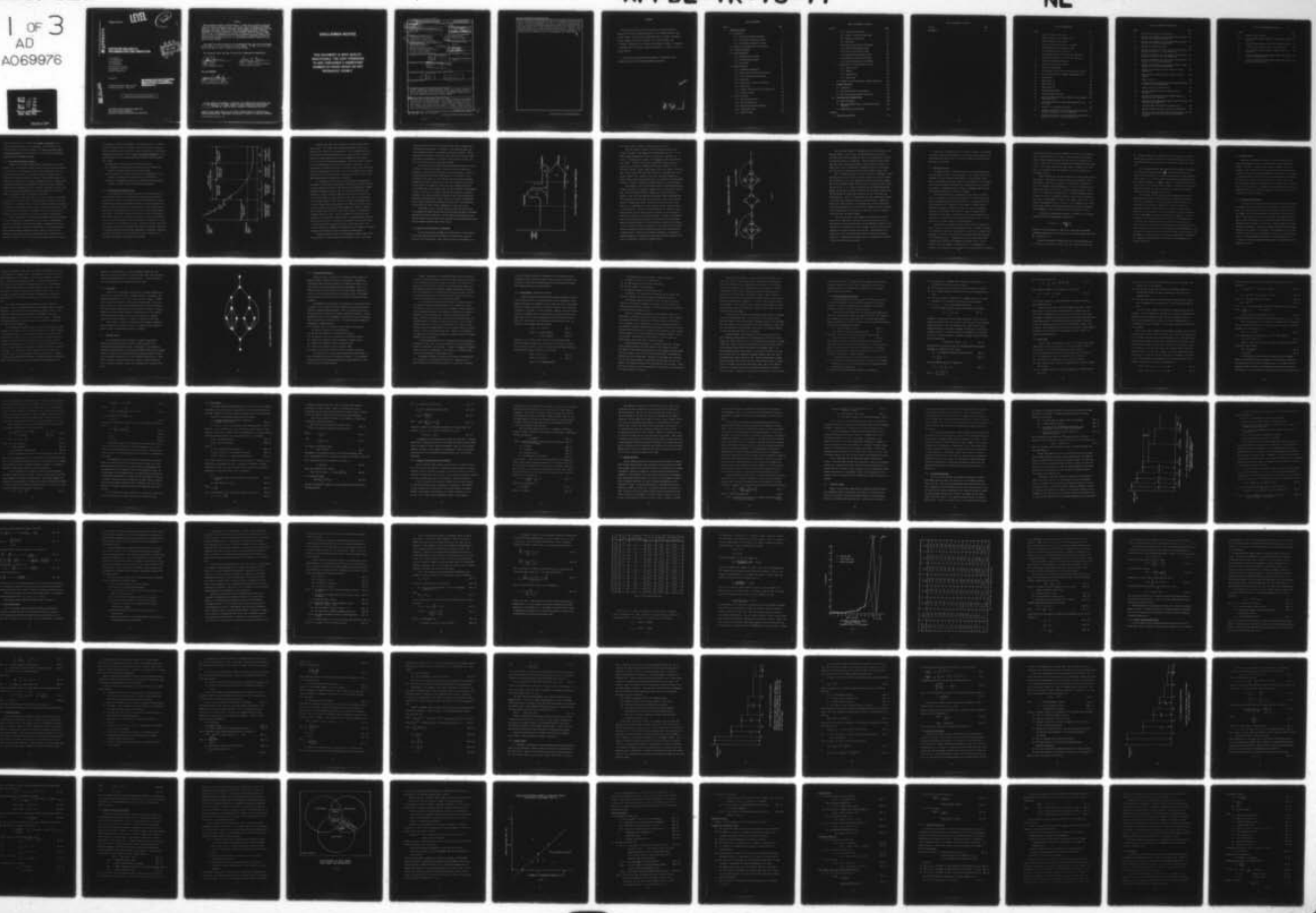
AD-A069 976 DAYTON UNIV OHIO F/G 9/2
SOFTWARE RELIABILITY: DETERMINATION AND PREDICTION.(U)
JUN 78 L S GEPHART, C M GREENWALD F33615-77-C-3072
UNCLASSIFIED AFFDL-TR-78-77 NL

1 OF 3
AD
A069976



AFFDL-TR-78-77

LEVEL II

2

SOFTWARE RELIABILITY: DETERMINATION AND PREDICTION

L. S. GEPHART
C. M. GREENWALD
M. M. HOFFMAN
D. H. OSTERFELD

UNIVERSITY OF DAYTON
MANAGEMENT SCIENCE
DAYTON, OHIO 45469

JUNE 1978

TECHNICAL REPORT AFFDL-TR-78-77
Final Report May 1977 – June 1978

Approved for public release; distribution unlimited.

AIR FORCE FLIGHT DYNAMICS LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

JOHN W. DAVISON
Project Engineer
Control Management Group

JAMES W. MORRIS, Chief
Control Systems Development Branch
Flight Control Division

FOR THE COMMANDER

JAMES R. STANLEY, COL, USAF
Chief, Flight Control Division
AF Flight Dynamics Laboratory

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify _____, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE/56780/1 February 1979 — 100

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFFDL-TR-78-77	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): SOFTWARE RELIABILITY: DETERMINATION AND PREDICTION	5. TYPE OF REPORT & PERIOD COVERED: Final Report, May 1977 - June 1978
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): L.S. Gephart, C.M. Greenwald, M.M. Hoffman, D.H. Osterfeld	8. CONTRACT OR GRANT NUMBER(s): F33615-77-C-3072
9. PERFORMING ORGANIZATION NAME AND ADDRESS: ~~Management Science~~, University of Dayton, 300 College Park Avenue, Dayton, Ohio 45469	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 24030235
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Flight Dynamics Laboratory, Air Force Systems Command, Wright-Patterson AFB, Ohio 45433	12. REPORT DATE: June 1978 13. NUMBER OF PAGES: 220
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software Reliability, Software Reliability Models, Error Models, Mathematical Models, Failure Rates, Maximum Likelihood, Error Analysis, Software Error Data, Data Acquisition, Errors Per Interval, Times Between Errors, Mean Time To Failure (MTTF), Mean Time Between Failures (MTBF).

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This study gives a comprehensive review of software reliability determination and prediction techniques and models. Each technique and model is discussed and evaluated as to its applicability to the software in a real-time, automatic digital flight control system. A total of seven techniques, nine empirical models, and fifteen analytical models are studied. Whenever possible the techniques and models have been applied to real software error data. The report is divided into three sections. Section I discusses software reliabil-

(continued on reverse)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

bility in general and then focuses on each of the techniques and models individually. It provides a preliminary evaluation of each model and partitions out four of the most promising approaches, which are then analyzed more thoroughly. Section II addresses the absolute necessity of gathering well documented software error data as well as the problems associated with its collection. It also provides references for a number of software error data sets Section III includes conclusions relative to the most attractive models, recommendations for the collection of software error data, and suggestions for future study.

FOREWORD

This report discusses the state-of-the-art in the modeling and analysis of software reliability relative to an automatic digital flight control system. The principal investigator was L.S. Gephart, Management Science Program, University of Dayton. The project engineer was J. Davison, AFFDL/FGLD. The research team included K. Fudge, C. Greenwald, M. Hoffman, and D. Osterfeld. Special thanks goes to E. Mykytka for his initial inputs.

The results were gathered under Contract No. F33615-77-C-3072, sponsored by the Air Force Flight Dynamics Laboratory.

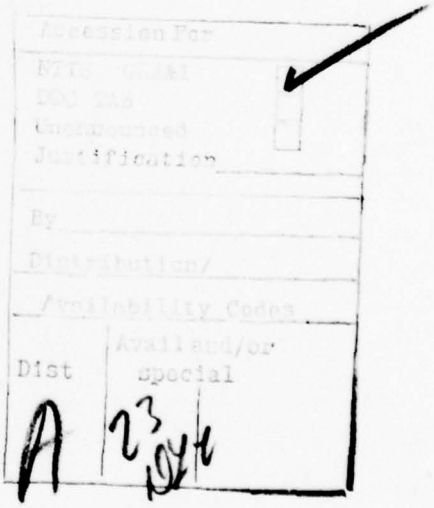


TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF ILLUSTRATIONS (Continued)

SECTION I

SOFTWARE RELIABILITY

1.1 INTRODUCTION

1.1.1 The Need for Reliable Software

It should be clear that reliable software is needed in an application such as an automatic digital flight control system. The success of the mission and the very lives of the crew are obviously dependent on the ability of the computer system to operate free of any catastrophic error. Hence, before we can entrust men and machines to such a system, we need a means of evaluating the reliability of the computer in terms of both hardware and software. Much work has been done in the area of hardware reliability, but, to date, relatively little has been accomplished in the area of software reliability. As a result, there is a need to develop techniques which will provide assurance of the reliability of a software package or, better yet, to develop a reliability model which will be suitable to predict the reliability of the software package at a given point in time. In this report we examine the techniques and the models which have been proposed to date and discuss their appropriateness and viability.

The high cost of software today is another reason why reliable software is important. For example, Boehm estimates that in 1972 the Air Force's expenditure on software was between $1 and 1.5 billion compared to $300-400 million on computer hardware [13]. By the late 1970's it is expected that software will represent at least 80 percent of the total outlay.

Even with large infusions of money, software reliability can be a real problem. Shelley [110] relates the following experience with the Apollo program:

"The world's most carefully planned and generously funded software program was that developed for the Apollo series of lunar flights. The

effort attracted some of the nation's best computer programmers and involved two competing teams... In the aggregate, about $600 million was spent on software for the Apollo program. Yet, almost every major fault of the Apollo program, from false alarms to actual mishaps, was the direct result of errors in computer software."

1.1.2 The Software Reliability Problem

Dickson et.al. [30] have defined software reliability as "the probability that a given software program operates for some time period, without a software error, on the machine for which it was designed given that it is used within design limits". We will loosely regard a software error as a failure of the program to perform as expected. These software errors can be categorized as either errors in design or implementation [92]. Design errors include misinterpretation of program specifications and incorrect problem formulation, while implementation errors include: typographical errors, logic errors, poor algorithm approximations, untested singularities or critical values, and misinterpretation of language constructions.

It is essential that we realize that a piece of software is unreliable if and only if it contains errors. Myers [86] states this point succinctly: "Although it is reasonable to want to express software reliability in relation to time, one should recognize that it is not really a function of time. Software reliability is a function of the number of errors, the severities and location of those errors and the way in which the system is being used."

Software reliability, then, differs from hardware reliability primarily in that software does not degrade with time as a result of environmental stress or fatigue (age). Because software does not degrade with time and since error correction eliminates the possibility of encountering an error again, software exhibits reliability growth. That is, once an error is detected and successfully corrected, the program is more reliable than it was before. Hence, the error or failure rate of software declines as errors

are corrected. Coutinho [22] emphasizes this important point: "It cannot be taken for granted that a deficiency, once discovered, will be fixed, or that the corrective action is efficient. Unless a deficiency is identified, isolated, and effectively corrected, there is no reliability growth". Pikul and Wojcik [92] list the following additional differences between hardware and software:

(1) Duplication of software does not introduce new errors.

(2) A comprehensive failure modes and effects analysis is impractical because of the large number of distinct logic paths.

(3) The correction of a software fault alters the configuration of the software and eliminates any possibility of its reoccurrence.

(4) There is no standardized approach for exhaustively testing software in order to assume it meets all operational requirements.

1.1.3 The Software Development Process

As a rough model of the general behavior of the software error rate over the software life-cycle, we propose the following four-phase model. (see Figure 1). Phase I is the design and initial testing period in which equations and algorithms are designed, flowcharts are prepared, modules are written, tested and integrated by the manufacturer. Errors encountered in this phase include typographical errors, syntax errors and the more blatant logic errors. This phase concludes with the end of initial system integration testing. Phase II is the "advanced" testing state where the program is tested as a whole, being subjected to inputs which are presumed to be typical of the inputs to be encountered by the user. This phase includes initial user testing and concludes with the software being placed in operation. Phase III is an operational phase in which limited maintenance is performed, correcting only those errors which it is cost-effective to remove. Phase IV is an operational stage where no maintenance is performed.

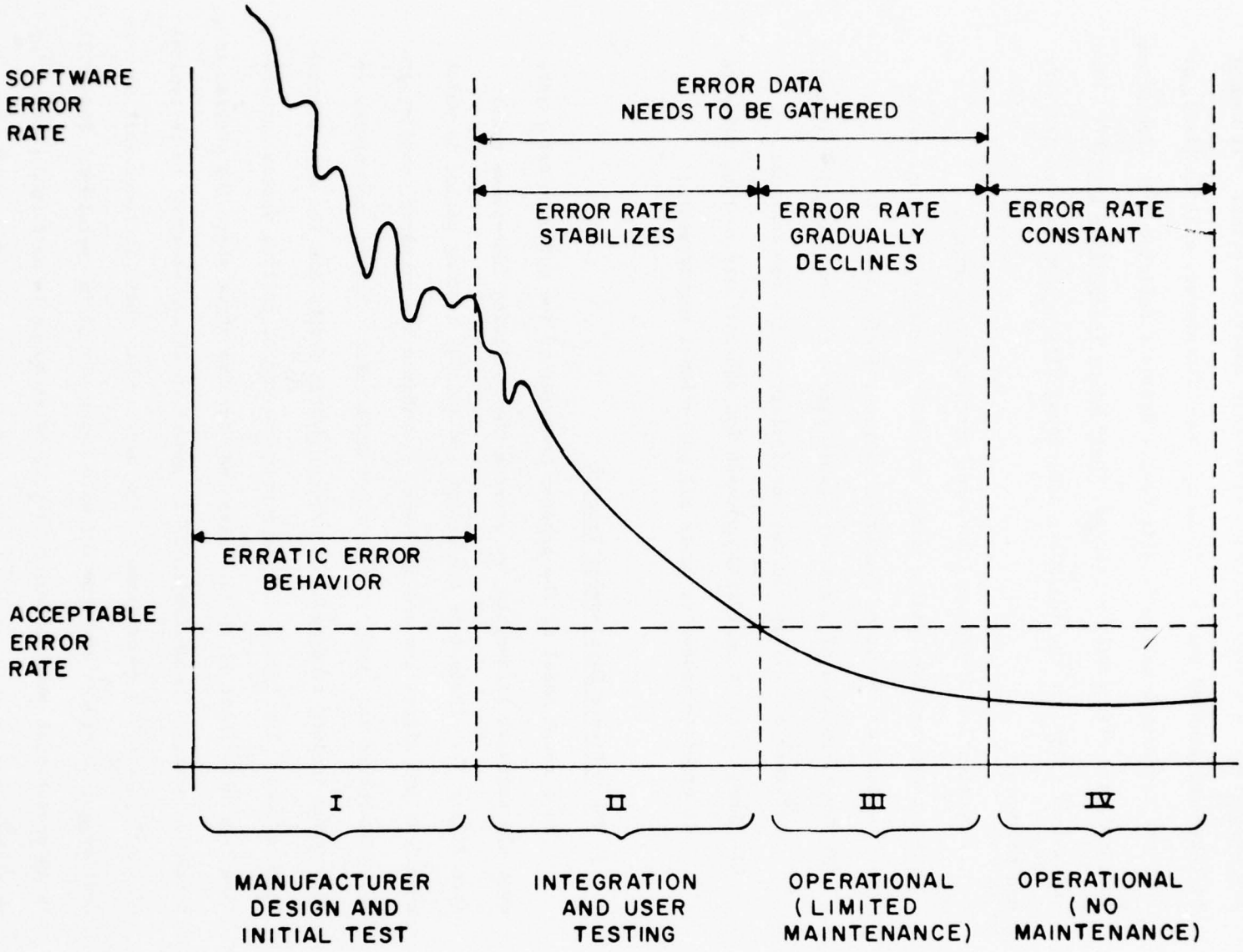


FIG. 1 – PHASES OF SOFTWARE DEVELOPMENT

In Phase I the error rate is expected to be quite high and may be quite erratic. As a result, data accumulated during this phase is not expected to be useful, if at all meaningful. As the integrated system is more extensively tested during Phase II, the occurrence rate of errors is expected to become smoother and decline as debugging effort continues (i.e. as errors are detected and eliminated). The error rate will continue to decline during Phase III and will in fact begin to level off. It is during Phase II and Phase III that we can obtain the most useful information (data) for developing an error model. It is in these phases that we can get a picture of the true error process. These are the critical stages in our model-building process and good data is required.

It is important to note here that we are implicitly assuming that the number of errors which is detected in a time interval and the collection of error counts over a series of time intervals can be modeled by a random variable and stochastic process. That is to say, the time between the detection of errors, the time between the detection and correction of an error, and the number of errors detected or corrected in a time interval are all random variables [109]. However, as Jelinski and Moranda point out [56], software errors do not appear to be random in any active sense; there is no physical mechanism for their generation. However, looking at their "passive" roles instead of their "active" roles, the assumption of randomness makes more sense. Errors are "passive" in that they can be regarded as a data/software interaction [67]. A program may work perfectly with data from a subset of the space of all possible inputs, but fail when a data point comes from a particular region of this input space. Since the data stream can be regarded as a random process it is natural to consider failures of the software as also constituting a random process.

In Phase IV errors tend to occur infrequently; however, these errors may be difficult to trace, identify, and eventually correct. Thus, some

errors may never be corrected either because they cannot be identified or cost-effectively eliminated. In addition, we can reasonably expect the error correction process to not always be completely successful. Indeed, the repair action may not successfully remove the error and may even cause new errors to be generated. As a result of these factors, we expect the error rate in Phase IV to be essentially constant.

Of course, we do not expect this error generation process to be local to Phase IV. Any time we attempt to correct an error we introduce the possibility of unsuccessful correction or new error generation. Belady and Lehman [9] describe this process with what they call a "model of fault penetration" (see Figure 2). They show that each time software undergoes attempted correction, a fraction E of total errors is removed (extracted) and new errors G are generated. Thus, after each attempted correction, a new composition of faults appears that consists of residual R and generated G errors. In this way, the authors show that the complexity, or unstructuredness of the system increases as repairs are made, thus increasing the difficulty of maintaining the system. Obviously the models and techniques which include a mechanism for handling the possibility of the introduction of new errors will have a distinct advantage over those that do not.

In Sections 1.2 thru 1.6 we will present, discuss, and evaluate all the techniques and models we have encountered which make an attempt to determine and/or predict software reliability. The discussion and evaluation will be carried out due to the fact that we are concerned with the software associated with an automatic digital flight control system.

1.2 COMPLETE TESTING AND PROOF OF CORRECTNESS

In this section we will consider the two ways which, at first glance, are the most "intuitively obvious" methods of certifying that a piece of software is true and accurate. Then, after more careful investigation, we

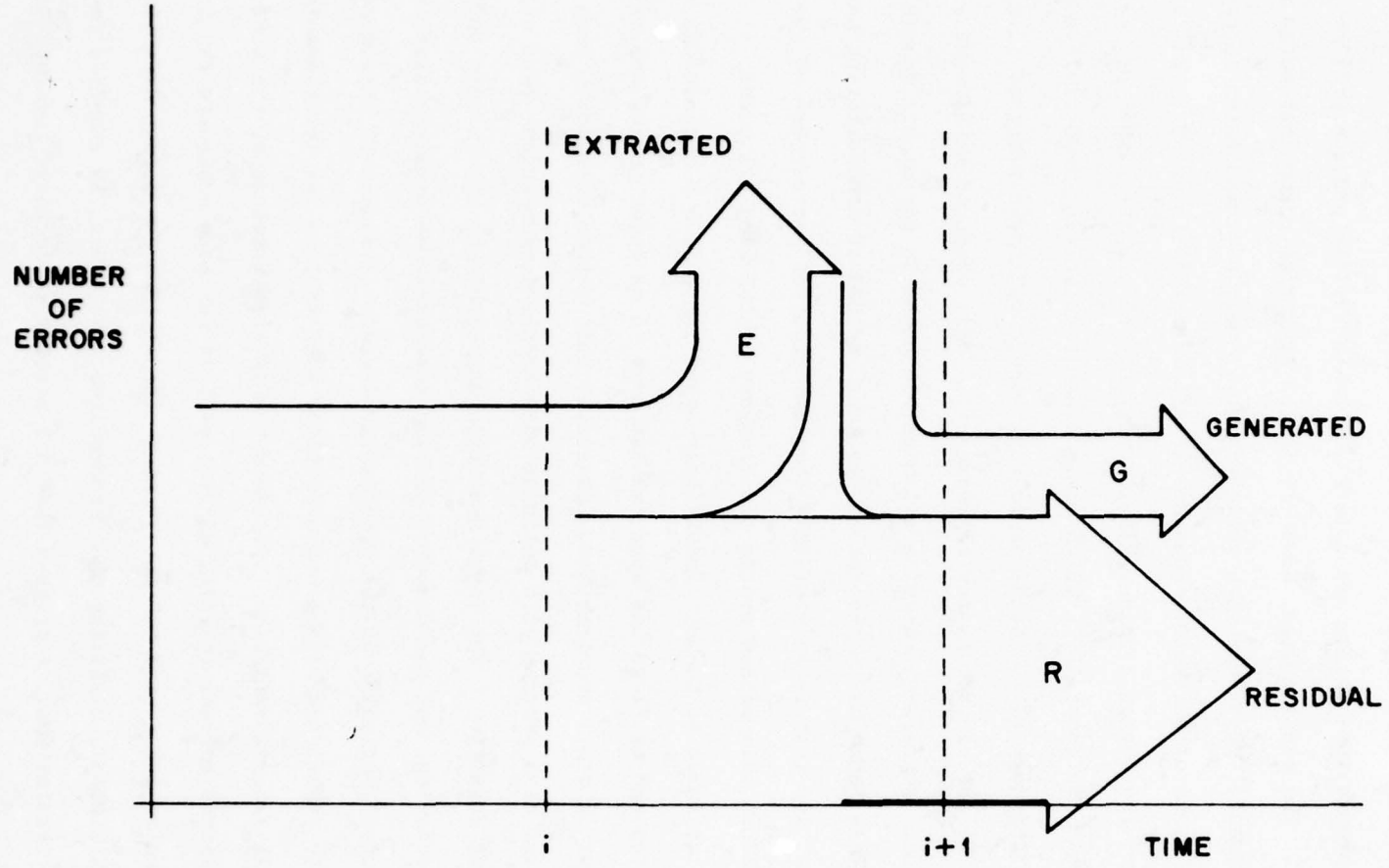


FIG. 2 - PRIMITIVE MODEL OF "FAULT PENETRATION"

will see why neither of them is really useful in our effort.

The first method is simply to check all possible logical paths through the program. This sounds very straightforward and clear. If all paths through the program are correct, then the entire program will be correct. The problem lies, however, in the number of paths that would have to be tested. There may very well be a great many of these paths in a typical piece of software. Consider the flowchart in Figure 3. This flowchart represents a rather simple program, and yet the number of possible logic paths is about 10^{20}, which makes exhaustive testing impossible even using a computer [15]. Needless to say, more complicated programs (e.g. digital flight control system software) will only compound an already absurd situation. Therefore, checking all possible paths is not at all feasible.

This leads us to the second "evident" method of demonstrating perfect software, and that is to perform a complete proof of correctness on the program. Since this method avoids the drudgery of checking all paths, it appears at first to be more intuitively appealing. Such a methodology was first proposed by Floyd and Naur and has come to be known as the assertion method or informal proof method. Basically this method associates each input line of a program with an input assertion and each output line with an output assertion. The input assertion expresses any constraints on the input variables, and the output assertion expresses the desired relation among output variables if the program terminates. A number of intermediate lines are associated with other assertions that express the relationship among all program variables. Provided the program terminates, the program can be proved correct (partial correctness) if for each procedure the input assertion together with the intervening code implies the output assertion. A separate proof, utilizing the intermediate assertions, is required for program termination. A program that has partial correctness together with proof of termination is said to be totally correct.

SIMPLE PROGRAM WITH LOGIC PATHS

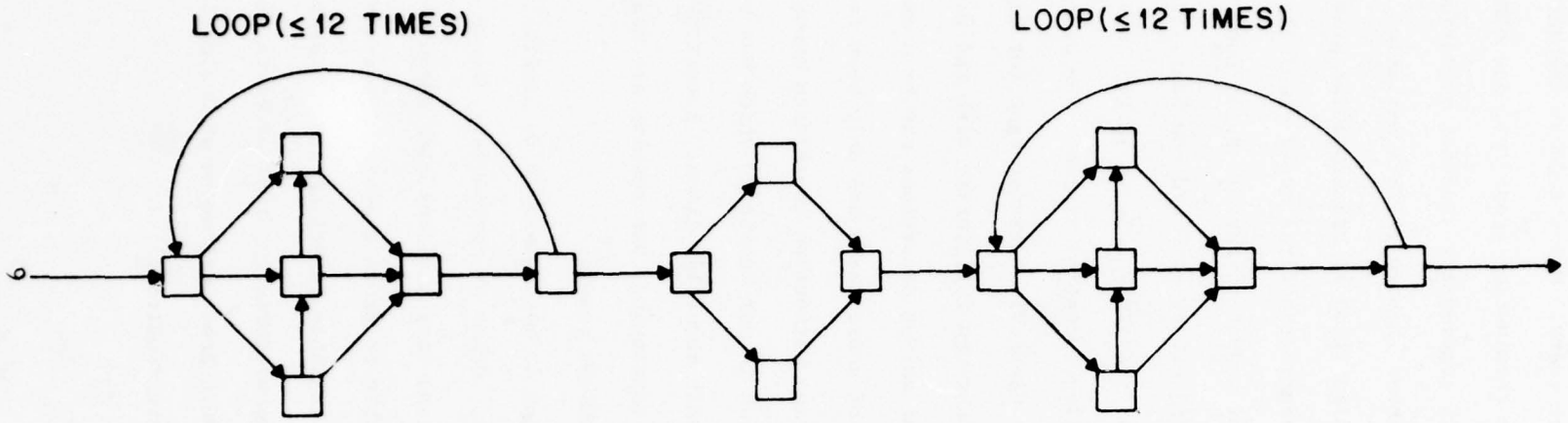


FIG. 3

There have been a number of implementations of and extensions to the assertion method in recent years. One extension combined the partial correctness proof and the termination proof into one single proof, but with no associated reduction in complexity. Another variation utilizes formal logic and certain correctness theorems. This variation, referred to as formal proof, relies on algorithms to transform the proof of correctness into the proof of a theorem in the first order predicate calculus [75].

While a significant amount of work is being done to implement, extend, and improve upon the initial effort of Floyd and Naur, the proof of correctness techniques are not as yet applicable to large scale software packages such as in an automatic flight control system. The process is long, tedious and error prone. It is a difficult exercise even for highly skilled programmers and requires a thorough familiarity with the program being proved. Just determining the input/output assertions can be a very arduous and challenging task. Proof of correctness has only been realistically applied to relatively short programs without an inordinate amount of complexity. Indeed, the longest program to our knowledge which has been proved using these methods contained only about 100 lines. Research into the automation of the program proofs is continuing, but results are still in a stage of early development and problems remain.

Proof of correctness is most amenable to application to logical properties and algorithms. Other properties(e.g. round off error) are more difficult to handle and many have not been adequately addressed as yet. Consequently, it is possible to prove a program "correct" but then find the program is not perfectly reliable in actual use. Misinterpretation of intended use cannot be demonstrated by proof of correctness techniques either [75]. Therefore more practical means of evaluating the reliability of large scale software are required.

While proof of correctness is not an adequate solution to the software reliability question we are pursuing, it has far more promise than the total testing mentioned earlier, and is an area where further research may prove beneficial in the future.

1.3 SEEDING AND TAGGING

An important aspect of software reliability is the determination of the total number of errors in a given program. Such a number, coupled with the number of errors discovered and removed to date, would give a good means of interpreting the present status of the reliability of the software. The user and/or developer could make a more informed decision as to whether the software is acceptable enough or whether the debugging should continue and how much time and effort needs to be expended.

Two possible methods of determining the total number of software errors in a program are "seeding" and "tagging". The peculiar names come from the standard wildlife-conservation practice of estimating the size of animal populations thru "tagging" a captured sample of animals from the population, releasing ("seeding") the sample back into the population, capturing a second sample, and counting the number of tagged animals. Assuming the second sample is representative of the entire population, it is possible to estimate the total population by comparing ratios. Feller discusses the technique applied to animal populations.

With respect to software there have been two variations proposed to date. The first is due to Mills and Hyman [138] and they suggest "seeding", i.e. deliberately inserting a certain number of errors into a program before debugging is to begin. The person doing the seeding should obviously not do the debugging as well. A number of errors will be discovered during debugging, some of which will presumably be "seeded" errors. Thus, an estimate of the total number of original errors in the program can be obtained. The problem with this procedure lies in the difficulty of inventing

and placing errors that are similar enough to the errors in the program so that the inserted errors are as likely to be discovered as the original errors. Without knowing anything about the errors in the program, this would be a difficult job and would introduce more variability into the final results. For a large software system the number of "seeded" errors would have to be quite large also, and placement of the errors becomes more problematic as well.

Rudner [119,120] has done the most work in this area and has made the most progress. She proposes a second variation of the technique, and her work appears to have the most promise. This method called "tagging" involves two debuggers working independently on a piece of software with an unknown total number of errors, N. The first debugger discovers t errors which can be considered the "tagged" bugs. The second debugger discovers s errors, which can be considered the sample. The two sets of discovered errors have a certain number of errors, c, in common. The ratio of c to s should be approximately equal to the ratio t to N, assuming that all errors have an equal chance of being discovered and no new errors are introduced during debugging. Thus, an estimate of N can be determined: $\hat{N} = \frac{st}{c}$.

A statistical, formal approach would be to consider c as a discrete random variable with a hypergeometric distribution:

$$p\,(c|s,t,N) = \frac{\binom{t}{c}\binom{N-t}{s-c}}{\binom{N}{s}}$$

The mean of this distribution is $\frac{st}{N}$ and it follows that the maximum likelihood estimator of N, call it N_o, will be the largest integer less than $\frac{st}{c}$. Note that $N_o \simeq \hat{N}$.

Rudner also develops a method whereby more than two debuggers can be utilized to obtain another estimate, N_{od}, with a smaller variance than

N_o. Confidence limits can be determined for N_o and N_{od}, the latter being narrower than the former as expected. Still another estimate, call it N_1, is proposed which not only has a smaller variance than N_o but also a lower bias.

The two assumptions mentioned earlier (equal probability of discovery of each error and the non-introduction of new errors) may possibly be unrealistically restrictive. The second assumption has not been eliminated from the model as yet. However, Rudner has developed a model modification for possible replacement of the first assumption. The modification basically divides errors into as many different categories of discovery difficulty as required and proceeds to estimate the number of errors in each category rather than just the aggregate error total N.

The techniques and estimators developed by Rudner appear to have enough promise and appeal to warrant further investigation and research. The various estimations of N have a much sounder basis than many of the other models we have studied. Furthermore, the extra capability of determining confidence limits makes the technique even more attractive. Application of the techniques to an actual debugging situation should prove most interesting from an evaluation and validation standpoint. It may be necessary, however, to modify the model to eliminate the assumption that no new errors are introduced during the removal of a discovered error. Another important consideration in the application of the techniques will be the increased cost of having two or more debuggers working independently. With only <u>one</u> test procedure, the cost of testing is already a major expense in the development of a piece of software, and it will be necessary to determine if the "tagging" techniques will be cost effective in the determination of software reliability.

1.4 STRUCTURAL MODELS

Most work on probabilistic modeling of software reliability has treated the program as a black box, and attempted to model its outward behavior. That work is described under Analytical Models, Section 1.6, following. This section discusses several efforts that have been made to look inside the box and incorporate knowledge of the structure of a program. It should be pointed out that all the structural models give an estimate of reliability for a particular software element at a specific time. They do not model the reliability improvements that occur over time. The general function of these models is to combine existing estimates of sub-routine or module reliability with structural information to devise an estimate of the reliability of the entire program.

1.4.1 Shooman Structural Model

In [112] Dr. M.L. Shooman describes a path oriented structural model. He assumes first that a program can be decomposed into a number of paths or cases. Which path or paths are executed is dependent on the data input to the program. It is also assumed that the identification of paths will be done at a high enough level to yield a relatively small number of cases, small at least with respect to the at least 10^{20} unique paths that can be identified in most programs. For each path, the model requires that three parameters be estimated. The paths are identified by N tests that between them will cause all paths to be executed. The parameters are t_i, the time required to run case i; q_i, the probability of an error on a run of case i; and f_i, the relative frequency with which the path represented by case i is executed. Using these parameters Shooman derives an expression for the system failure rate:

$$Z_o = \frac{\sum_{i=1}^{N} f_i q_i}{\sum_{i=1}^{N} f_i \left(1 - \frac{q_i}{2}\right) t_i} \qquad \text{(Eq. 1)}$$

This is the average probability of failure on one test case, over the mean run time for a single case.

In his paper, Dr. Shooman admits the massive difficulties in estimating the parameters. With extensive special code in the program, actual runs can capture data for relative frequency and run times. Alternatively he suggests assuming uniform usage of paths and use of an average run time. He makes no useful suggestions regarding q_i, the probability of failure for each path.

Halstead [49] attempts to relate various aspects of computer software to E, the number of elementary mental discriminations needed to produce the program. E is considered an accurate measure of the effort expended. He computes E from such factors as operator and operand vocabulary and program length. Funami [41] recently applied software physics to debugging data reported by Akiyama [1]. Funami reports an excellent correlation between E and the total number of bugs present for the nine modules described by Akiyama. If a precise relationship can be discovered, software physics may allow the direct estimation of Shooman's parameter q_i, the path error probability. The availability of such a technique is many years away however, and methods will need to be developed to account for variations in language and between programmers.

1.4.2 Littlewood Markov Model

Littlewood [69] proposes a model that more accurately reproduces program structure. Each section of coding in a particular program is identified as a state. The execution of the program can then be considered a series of transfers from state to state. If the probability of each transfer

occurring is independent of how long the system has been running, the system can be simulated by a Markov process. The parameters for this are N states and N^2 probabilities of the form q_{ij}. These are the probability that when in state i the system will transfer to state j. In order to include those errors that occur in interfaces Littlewood introduces λ_{ij}, the probability of a failure occurring during the transfer from state i to state j. The probability of an error occurring while executing in state i is ν_i. In his paper, Littlewood outlines techniques for determining the probability distribution for time to failure as long as the λ_{ij}'s and ν_i's are very small.

In a later paper Littlewood [70] introduces several additional parameters. F_{ik} is a probability function that determines, before a transfer from state i to state k, how long the program will remain in state i. This changes the transition process from Markov to semi-Markov. The other additional parameter is y_i, the cost of a failure in state y. Littlewood then suggests methods for calculating the cost of running the program and the overall failure probability.

As with Shooman's structural model, the level of detail depicted and consequently the number of states must be selected so as to limit the amount of computation required. If a system with 25 modules or sub-routines were being modeled at a module level, more than 1300 numeric parameters would be required. Again, the structural parameters, the q_{ij}'s and F_{ik}'s could be estimated by adding many counters and other special code to a program. The effect of so many changes on the program's reliability and timing is questionable however. As with Shooman's model, Funami's work may in time allow estimation of the ν_i's, the probability of failure in each state. Because the states can be identified with modules or subroutines, unlike in Shooman's model, it may be possible to use data from early in development when the

modules are tested separately. It must be remembered though that this model does not project any growth in reliability with time. Both Littlewood models are extremely difficult to solve numerically. For these reasons we feel further study would not be productive at this time.

1.4.3 Green Model

Thomas Green has developed a model [46,47] of program debugging which can be used to estimate the effect of program structure on debugging. The program is described as a set of nodes, arcs, and loops (Figure 4). Parameters describe the probability that when the program reaches a particular node each arc will be taken, and how many times each loop will be executed. The nodes are branch points and the arcs are sections of coding. A computer simulation then can randomly distribute errors among the arcs and simulate execution with various inputs. This can reveal the effect of program structure on the effectiveness of a sequence of randomly selected tests.

Green's results to date have confirmed several intuitive assumptions about the difficulty of testing all parts of a program and the tendency of errors to "hide" in little-used sections of a program.

1.5 EMPIRICAL MODELS

In the following sections, we present a number of approaches to software reliability which do not attempt to model the actual error detection/collection/generation process. These models do not hypothesize a particular hazard function or a particular reliability function, but attempt, instead, to either select a particular function based upon empirical data or relate reliability statistically to certain software characteristics. These so-called "empirical models" approach the problem of software reliability from a statistical viewpoint, rather than a mathematical or analytical viewpoint.

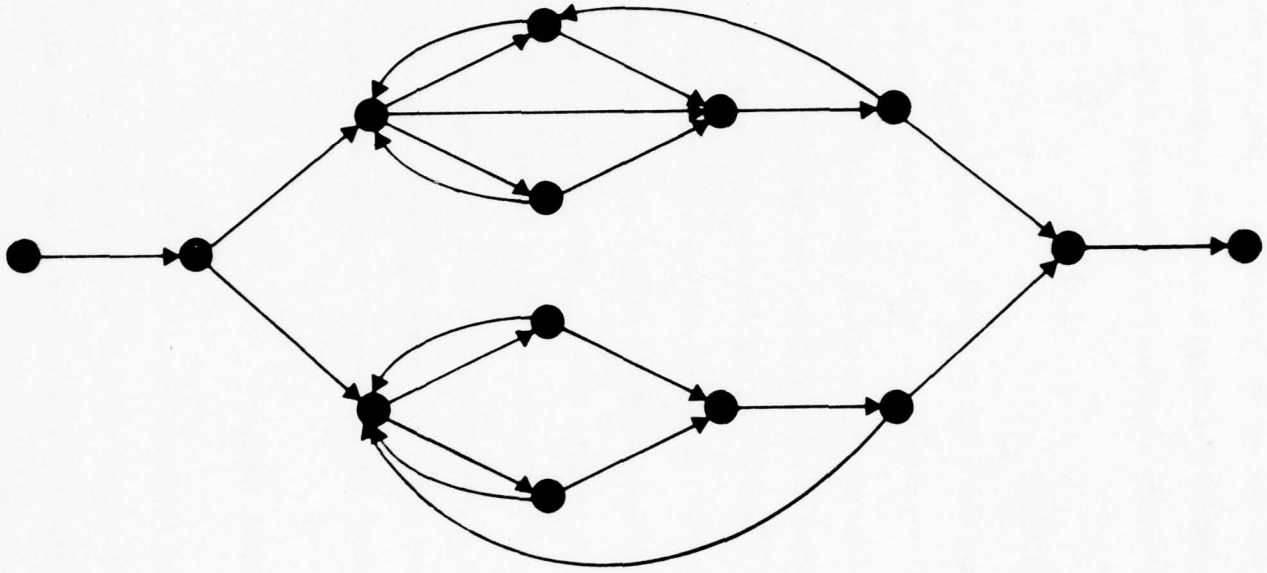


FIG. 4 - DIRECTED GRAPH REPRESENTATION OF A PROGRAM

1.5.1 Schneidewind Methodology

A major objective of statistical or empirical software models is to predict software reliability based on test program data. Specifically, Schneidewind [108] investigates methods for estimating the amount of test time required to satisfy program reliability requirements. Then, a reliability function can be obtained if a theoretical density function f(t) can be fitted to the empirical data of time between errors. Furthermore, confidence limits may be derived for the theoretical reliability function parameters.

Schneidewind also has proposed the utilization of statistical testing techniques. First, analysis-of-variance (ANOVA) may be used in testing for differences between and among programs. Hence, the results may be used to identify the major contributors of program reliability variability. Second, goodness-of-fit tests can be implemented to specify the type of distribution which is applicable to software failures. The general approach proposed by Schneidewind is summarized below:

(1) Tentatively select a reliability function based on the shape of the frequency function of the empirical data,

(2) Estimate the parameters of the reliability function,

(3) Use goodness-of-fit tests to identify the reliability function,

(4) Estimate the reliability function parameters confidence limit,

(5) Estimate the reliability function confidence limit,

(6) Predict the reliability and its confidence limit,

(7) Compare the predicted reliability with the required reliability.

This appears to be a very reasonable approach and could perhaps be used in conjunction with one of the analytical models. It is intuitively appealing because of its logical approach and its applicability to a very general class of reliability models.

However, implementing the aforementioned approach may be complicated by the fact that the time between errors or the number of errors per interval is not a stationary process with respect to test time. As a result of the reduction in the error rate, the form of the distribution may change, or it may remain the same with its parameters changing. Nevertheless, in either case, this indicates that a reliability function which is based on the total set of data points may not be an accurate predictor because the entire set of data is not representative of the current error process.

If the distribution remains the same during the testing process and the parameters change, Schneidewind suggests using a smoothing technique with the most recent data points to derive new parameter estimates relative to the upcoming time interval. Thus, parameter estimates would continue to be updated as the testing continues. However, if the distribution does change, a more serious problem arises, that being the identification of the distribution which is relevant for each stage of testing. Once a suitable reliability function is obtained it becomes possible to estimate the lower confidence limit for the MTBF and thus use this estimate with the reliability function to obtain the reliability lower limit. Finally, the estimate may be compared with the specified requirements to determine whether an acceptable reliability has been achieved.

As was previously mentioned, the reliability function of one stage may not apply at a future time. It is then assumed that the revised reliability function is applicable at the next stage. A change in the <u>type</u> of reliability function is made at the completion of a test stage, if a noticeable change occurs in the distribution of time between errors.

Schneidewind concludes his discussion by noting the following points: (1) the specifics of the approach will probably be supplemented by an improved model which is now under development, (2) reliability is estimated

to satisfy program reliability requirements, and (3) important factors affecting software reliability and its accuracy include the difference of reliability characteristics among programs and the previously discussed non-stationary process of error occurrence.

1.5.2 Weibull Model (Proposed by Coutinho)

A general hardware oriented reliability growth technique to predict future system failures is to utilize past data and fit parameters of the Weibull distribution as suggested in [22], [123], and [124]. This approach models the software error rate as an explicit function of time and not as a function of the number of errors remaining. Coutinho notes "when errors, failures or stoppages occur, the causes must be found, corrective action taken, and the test re-run to demonstrate that the corrective action is effective". Specifically, the instantaneous failure rate at time t is given by the two parameter density function:

$$Z(t) = (\eta/\sigma)\ (t/\sigma)^{\eta-1} \qquad \text{(Eq. 1)}$$

where η = a shape parameter (Def. 1)

σ = a scale parameter (Def. 2)

Coutinho, in [22], further states "the error rate declines rapidly during the debugging period until it levels off and reaches a residual error rate which continues to exist in large, complex debugged programs". The MTTF and reliability equations are as follows:

$$MTTF = [\sigma\Gamma(1/\eta)]\ /\eta \qquad \text{(Eq. 2)}$$

$$R(t) = \exp\ [-(t/\sigma)^{\eta}] \qquad \text{(Eq. 3)}$$

where Γ is the gamma function.

The following inputs are necessary to utilize this model.

(1) The number of errors in each time interval.

(2) The time between each time interval (CPU Time).

(3) The total number of time intervals.

(4) The cumulative number of errors found to date.

Estimators of the shape and scale parameters may be derived using plotting techniques, the method of moments and maximum likelihood.

This approach seems reasonable due to the wide range of possible shapes that the Weibull distribution can assume. Also, the Weibull distribution does reflect our assumption of reliability growth and is probably the most relevant distribution to use.

Thus, this technique was applied to three different data sets to determine the usefulness and accuracy of such an approach. First, Coutinho [22] applied the Weibull technique to the "army field test data" and his results (graphs) are very questionable. The data set under consideration was 36 weeks in duration, however his results appear to be based on the last 19 weeks (weeks 18-36) solely. He seems to have discarded the first 17 weeks from his analysis, yet he doesn't present any supporting evidence as to why this was done. Nevertheless, the line "fitted" to his data cannot be derived unless only the last half of the data is used.

A second question pertaining to his analysis also arises. The data points Coutinho plotted on his graph (reference [22], Figure 1) were not the number of errors discovered per week but was the cumulative average number of errors per week. Thus, the initial points plotted will be weighted more heavily than later occurring ones. Naturally, as the sample size increases, the error data (actual number of errors discovered per week) will have less impact on the cumulative average and any fluctuations will be minimal. Finally, when the error data is plotted on log-log paper (3x3 cycles) any deviations will be extremely difficult to observe.

After observing Coutinho's results, we applied this method to two other sets of data, First, Sukert's data [124] was analyzed and grouped into weekly intervals prior to application of this model. On the log-log paper, the points plotted are reasonably similar to Coutinho's in that the initial points exhibit erratic behavior and only upon reaching the 12th week of the test period does a trend begin to appear. Thus, we would be forced to throw out the first 11 weeks of error data which we cannot justify. However, we continued and attempted to fit a line to the data points (both actual number of errors per week and the cumulative average number of errors per week) using simple linear regression. Again, in this application, the results were not encouraging.

The other set of data that we applied this Weibull method to consisted of only seven data points [30]. In [22] Coutinho states that you need a sufficient number of data points before a meaningful analysis can be prepared. Due to the small data set it was difficult to clearly identify a trend. However, the regression line derived was a much better fit than those obtained in the two previous applications of this method.

Coutinho comments on his application "the initial portions of the $\Sigma D/t$ (cumulative average number of errors detected per week) curve for both samples are similar and do not plot on a straight line as expected, but exhibit a wave form. This is caused by peculiarities of the test plan which results in deficiencies being initially discovered in an irregular order". The above statement may have some validity to it; however, we must also consider the fact that due to the cumulative averaging technique, early data points are weighted more heavily than later ones. Thus, it would appear as though even with the most erratic error data one could eventually project a "fitted line" as the sample size becomes sufficiently large. Due to this apparent deficiency, we do question the use of cumulative averaging in software reliability studies. Furthermore, Coutinho fails to mention why only weeks 18-36

were used in his analysis. In light of all the problems and questionable procedures observed, we feel that such a technique will not serve as an appropriate predictor in analyzing the software portion of an automatic digital flight control system.

1.5.3 Corcoran-Weingarten-Zehna Model

A model developed in [21], more applicable to hardware, focuses on the observed performance of an item and hence estimates the current reliability. The approach presented differs considerably from the models developed exclusively to predict software reliability.

Initially it should be noted that this particular model ignores the time of the test and only considers the outcome of N trials. This appears to be a limiting factor on the applicability of the proposed model; however, further investigation is in order. Below are some of the symbolic definitions given by Corcoran, Weingarten and Zehna:

N = the total number of tests (Def. 1)

N_o = number of observed successes in N tests (Def. 2)

Hence, the current reliability after N tests may be given as N_o/N.

$N-N_o$ = the number of failures in N tests (Def. 3)

Now, a primary objective is to identify some of these failures, take corrective action, and calculate the new reliability. It was proposed that after corrective action had been taken N' additional tests be conducted with N'_o being the number of additional successes noted. Finally, a new reliability estimate may be computed by N'_o/N'.

A listing and evaluation of the model assumptions is presented below.

(1) It is assumed that a given test may result in success, with an unknown probability p_o or one of K failure modes, with the unknown probability of failure type i given as q_i.

(2) The N tests to be performed are assumed to be independent.

(3) By fixing K, it is assumed that no additional failure modes are generated when corrective action is taken.

(4) No corrective action is taken until all N tests have been executed.

(5) It is assumed that when a failure occurs it can be categorized as to type.

(6) If a type i failure is observed, it is assumed that there is a known probability a_i of removing that failure by corrective action.

(7) $a_1, a_2, \ldots\ldots, a_k$ are conditional probabilities.

The first assumption seems reasonable and implies the following:

$$p_o + \sum_{i=1}^{k} q_i = 1.0 \qquad \text{(Eq. 1)}$$

Assumption 3 is the first one which seems questionable in that there is a distinct possibility that the corrective action taken may in fact introduce additional failures. The sixth assumption also causes concern since a known probability of correction is assumed. Realistically, it would seem extremely difficult to accurately quantify the probability of successful correction. Finally, assumption 7 presents no problems statistically and may be interpreted as:

$$P(\text{correction} \mid \text{failure}) = a_i \qquad \text{(Eq. 2)}$$

Since failures are defined by type, the following should prove helpful throughout the remainder of this section.

N_i = the number of observed failures of type i. (Def. 4)

$$N_o + \sum_{i=1}^{k} N_i = N \qquad \text{(Eq. 3)}$$

Current reliability may now be computed as:

$$p_o^* = p_o + \sum_{i=1}^{k} y_i q_i \qquad \text{(Eq. 4)}$$

where $y_i = \begin{cases} 0 & \text{if } N_i = 0 \\ a_i & \text{if } N_i > 0 \end{cases}$

Then, the mean reliability is:

$$\mu = p_o + \sum_{i=1}^{k} a_i q_i \left[1-(1-q_i)^N\right] \qquad \text{(Eq. 5)}$$

Then, using the estimates of the parameters in equation 4:

$$p_1 = N_o/N + \sum_{i=1}^{k} a_i N_i/N \qquad \text{(Eq. 6)}$$

In [21] other estimators are presented similar to equation 6 and thus, are not presented here.

Finally, Corcoran et.al. state "with regard to our model, we feel that we must stress the obvious and remark that our estimators are not better than the probabilities (a_i) of successful corrective action which are assumed". Furthermore, they mention that their proposed model does have limited applicability. And although this model does predict reliability it does not estimate the number of errors or times between errors. Thus, in light of these limitations and the fact that test time is totally ignored, we believe that such a model will not address the overall software reliability problem adequately.

1.5.4 Weiss Model

The Weiss model was developed to predict a mean time between failure (MTBF) curve, based on a given number of trials, and compare it with the actual MTBF's. The following assumptions are given in [126].

(1) The exponential distribution is assumed for the times to failure.

(2) It is assumed that there are M possible sources of failure.

(3) The failure rate is different for each source of failure.

(4) When a failure occurs, there is a probability ($P_c < 1$) that the failure is corrected.

(5) It is assumed that MTBF for the system changes by a constant percent each trial.

(6) For each of the M sources of failure, this type of error may occur several times in the software.

(7) For some error types, once the error is discovered, there is a high probability that all other occurrences of this type will be removed at once.

(8) For other error types, a relatively small fraction of their occurrences elsewhere in the software will be detected and corrected at once.

The first three assumptions seem fairly sound and do not pose any problems. However, assumption 4 requires that each P_c be estimated from previous data. It would appear that subjective judgment would play a major role in establishing the various P_c values. Quantifying P_c's is a formidable task and is similar to the process we question in Corcoran's et. al. model.

Weiss states that a standard test program is developed, and the system is tested until it fails (time to failure = T_m) or until some maximum time (T_t) is reached when the test is then terminated. After the completion of testing, design changes are made and incorporated into the system, and the testing process resumes. Then after n systems have been tested, n_f have been terminated by failure (T_m), and n_t are terminated when a specified operating time (T_t) is reached. Finally, it has been assumed that MTBF, [T(i)], and its reciprocal a(i) are functions of the number of the trial, i, and certain parameters that depend on the reliability growth process assumed. Symbolically, this may be interpreted as:

$$T(i) = T(i, \alpha_1, \alpha_2, \ldots\ldots\ldots\ldots, \alpha_m) \qquad \text{(Eq. 1)}$$

$$a(i) = a(i, \alpha_1, \alpha_2, \ldots\ldots\ldots\ldots, \alpha_m) \qquad \text{(Eq. 2)}$$

Now, the density function for the sample resulting from the test of n systems is:

$$\phi = [\prod_{n_f} a(i)]\ \Delta^{-n_t}\ \exp\ [\ -\sum_{n} a(i)\ T_m(i)] \qquad \text{(Eq. 3)}$$

or

$$\ln \phi = \sum_{n_f} \ln a(i) - \sum_{n} a(i)\ T_m(i) - n_t \ln \Delta \qquad \text{(Eq. 4)}$$

where Δ = a very small increment of time (Def. 1)

$$a(i) = T(i)^{-1} \qquad \text{(Def. 2)}$$

If $\alpha_1, \alpha_2, \ldots, \alpha_m$ are determined via maximum likelihood the set of equations $\frac{\partial \ln \phi}{\partial \alpha_j} = 0$ are obtained.

$$\sum_{n_f} \partial \ln a(i) / \partial \alpha_j - \sum_{n} T_m(i) \partial a(i) / \partial \alpha_j = 0 \qquad \text{(Eq. 5)}$$

The above equation may be applied to obtain the M.L.E. of $\alpha_1, \alpha_2, \ldots\ldots, \alpha_m$.

Weiss then presents the case where it is assumed that the probability that the system will fail in Δt is $a\Delta t$ with a being a constant. Then the probability of no failure in t is:

$$\phi(t) = (1 - a\Delta t)^{t/\Delta t} \qquad \text{(Eq. 6)}$$

As Δt approaches zero, equation 6 goes to e^{-at}. Thus, MTBF is:

$$T = \int_o^\infty te^{-at}\ a\ dt = a^{-1} \qquad \text{(Eq. 7)}$$

Hence, if there are M sources of failure, then the general form of equations 6 and 7 becomes:

$$\phi(t) = \exp\ [-\sum_{m} a_j\ t] \qquad \text{(Eq. 8)}$$

$$T = (\sum_{m} a_j)^{-1} \qquad \text{(Eq. 9)}$$

In the application made by Weiss "unexpectedly close agreement between the derived curve and the original can be reasonably explained on the grounds that this is only one example; that if the example had been repeated a number of times, poorer agreement would have been obtained on the average". It is not clear as to whether the data Weiss used was actual

error data or data generated through simulation. Thus, the true utility of this model is not known, and although others have not actively pursued this approach, it may require further investigation.

1.5.5 Weibull Model (Proposed by Wagoner)

The negative exponential distribution, probably the most widely used probability function in reliability analysis, is briefly discussed by Wagoner [134]. He states that one assumption of this distribution is that the failure rate is constant which greatly restricts its applicability. Furthermore, it is implied that the probability of failure is solely dependent on the length of the time interval and independent of what hour of operation the unit under consideration is in. This does not appear to be true, especially in the initial phases of a program, when software errors occur quite frequently and the operating times are relatively short. Thus, a distribution with a decreasing failure rate would appear to be more appropriate.

The Weibull distribution, due to its allowance for an increasing, decreasing, or constant failure rate satisfies the aforementioned criteria. Wagoner also cites Coutinho and Hudson as having applied the Weibull distribution to model the software error detection process. However, these authors used calendar time as the independent variable in their analyses. This approach (supported by four applications to data) yielded values for the shape parameter in excess of unity. Hence, this would imply that the failure rate increases with time. Wagoner states that if this were the case, programs would never achieve operational status since the number of software failures per unit time would increase the longer the program was run. Thus, he hypothesizes that choosing CPU time as the independent variable would be a better choice. Supporting this statement is the fact that calendar time does not account for the significant increase in running time per week which often

occurs as a program nears operational status. Secondly, calendar time will not reflect the time periods when a program is not run at all. Thus, CPU time should be a better measure than calendar time. The forthcoming approach, proposed by Wagoner, was applied to a set of data and in that particular case, produced good results. However, Wagoner carefully notes that although the Weibull distribution yielded accurate results in this one application, similar results might not be obtained with other sets of data.

Below, equations 1 and 2 give the probability density function and the cumulative distribution function respectively:

$$f_T(t) = \begin{cases} \frac{\eta}{\sigma} \left(\frac{t}{\sigma}\right)^{\eta-1} \exp\left[- \left(\frac{t}{\sigma}\right)^{\eta} \right] \\ 0 \quad \text{elsewhere} \end{cases} \qquad \begin{cases} t \geq 0 \\ \sigma > 0 \\ \eta > 0 \end{cases} \qquad \text{(Eq. 1)}$$

$$F_T(t) = 1 - \exp\left[- \left(\frac{t}{\sigma}\right)^{\eta} \right] \qquad \text{(Eq. 2)}$$

where σ = a scale parameter (Def. 1)

η = a shape parameter (Def. 2)

t = the cumulative debugging time (Def. 3)

$F_T(t)$ = the cumulative fraction of errors detected at time t (Def. 4)

Thus, it may be observed that for $\eta>1$ the failure rate increases with time, for $\eta<1$ the failure rate decreases, and for $\eta=1$ the failure rate is constant. For the third case ($\eta=1$) the Weibull distribution reduces to the negative exponential distribution, which was previously discussed.

If it is assumed that the cumulative number of failures (errors) discovered is distributed as a Weibull function, then the actual error data may be compared to the model predicted values. Now, Wagoner suggests implementing a transformation to linearize the cumulative distribution function in order to simplify the parameter estimation process. Initially, the cumulative distribution function may be restated as:

$$1 - F_T(t) = \exp\left[- \left(\frac{t}{\sigma}\right)^{\eta} \right] \qquad \text{(Eq. 3)}$$

$$\frac{1}{1 - F_T(t)} = \exp\left[\left(\frac{t}{\sigma}\right)^{\eta}\right] \qquad \text{(Eq. 4)}$$

Then,

$$\ln \ln \left[\frac{1}{1 - F_T(t)}\right] = \eta \ln t - \eta \ln \sigma \qquad \text{(Eq. 5)}$$

or, the equation for a straight line

$$y = mx + b \qquad \text{(Eq. 6)}$$

Defining the variables in equation 6 in terms of the Weibull distribution gives:

$$y = \ln \ln \left[\frac{1}{1 - F_T(t)}\right] \qquad \text{(Def. 5)}$$

$$m = \eta \qquad \text{(Def. 6)}$$

$$x = \ln t \qquad \text{(Def. 7)}$$

$$b = -\eta \ln \sigma \qquad \text{(Def. 8)}$$

Utilizing the method of least squares in conjunction with the software error data gives estimates of the model parameters η and σ. Then a fitted line may be derived and plotted on semi-log paper. Cumulative CPU time represents the x-axis and the detected error function corresponds with the y-axis.

Although Wagoner states that the Weibull distribution successfully models the particular case that he analyzes, there do appear to be some unanswered questions. First of all, with the data set that Wagoner used, he assumed that the total number of errors in the program was known. We strongly question this assumption and are concerned with the applicability of the Weibull distribution in the event that the total number of errors is not known or cannot be accurately estimated. Secondly, whereas the Weibull distribution did produce favorable results in Wagoner's case, the overall utility of this approach cannot be assessed until additional applications are made.

At this time, we find some assumptions to be quite questionable and will focus our attention on models that appear to be more promising.

1.5.6 Nelson Model

This model, developed at TRW, examines the actual output of a program and compares it with the desired output with the resulting deviation being recorded. Based on this type of approach reliability may then be predicted.

Initially, the following definitions are presented.

p = a computer program which gives a computable function F on the set E (Def. 1)

$E = E_i$ for i = 1,2,.....,N (values of the input variables) (Def. 2)

Thayer et. al. in [128] then note that executing p produces $F(E_i)$ for each input E_i. Hence, due to imperfections in implementing p, p actually specifies a function F' which differs from the desired function F. Therefore,

$F(E_i)$ = the desired output (Def. 3)

$F'(E_i)$ = the actual output (Def. 4)

Then, if

Δi = an acceptable output tolerance (Def. 5)

$|F'(E_i) - F(E_i)| \leq \Delta i$ (acceptable deviation) (Eq. 1)

$|F'(E_i) - F(E_i)| > \Delta i$ (unacceptable deviation) (Eq. 2)

It is further stated that for equation 2, the E_i comprise a subset E_e of E, producing the unacceptable results. In addition to equation 2 yielding unacceptable output, it is also possible that (1) execution may terminate prematurely, or (2) execution may fail to terminate.

If,

P = probability that a run will result in an execution failure (Def. 6)

n_e = the number of E_i in E_e (Def. 7)

Then,
$$P = \frac{n_e}{N} \quad \text{(Eq. 3)}$$

Let R = the probability of no execution failures in a run of p (Def. 8)

$$R = 1 - P = 1 - \frac{n_e}{N} \quad \text{(Eq. 4)}$$

It should be noted that equations 3 and 4 are based on the assumption that each E_i is equally likely. However, the authors state that in operational use, the inputs are most probably not equally likely. Instead, they may be chosen in conjunction with some operational requirement.

The P may be expressed in terms of p_i (a probability distribution) by defining an execution variable y_i.

$$y_i = \begin{cases} 0 \text{ if the run with } E_i \text{ produces acceptable results} \\ 1 \text{ if the run with } E_i \text{ yields an execution failure} \end{cases} \quad \text{(Def. 9)}$$

Hence, $$P = \sum_{i=1}^{n} p_i \, y_i \quad \text{(Eq. 5)}$$

and $$R = \sum_{i=1}^{n} p_i (1-y_i) \quad \text{(Eq. 6)}$$

If n runs are performed, we have:

$$R(n) = R^n (1-P)^n \quad \text{(Eq. 7)}$$

Furthermore, if the inputs are chosen in some definite sequence, then

p_{ji} = the probability that E_i is chosen as the input on the $j^{\underline{th}}$ run (Def. 10)

So the probability that run j results in a failure is

$$P_j = \sum_{i=1}^{n} p_{ji} y_i \quad \text{(Eq. 8)}$$

Thus, the reliability for n runs is:

$$R(n) = (1-P_1)\,(1-P_2)\ldots\ldots(1-P_n) = \prod_{j=1}^{n} (1-P_j) \quad \text{(Eq. 9)}$$

or in exponential form:

$$R(n) = \exp \sum_{i=1}^{n} \ln(1-P_j) \quad \text{(Eq. 10)}$$

R(n) may be expressed in terms of t, the execution time, by utilizing the following approach.

Let Δt_j = execution time for run j. (Def. 11)

$$t_j = \sum_{i=1}^{j} \Delta t_i \quad \text{(cumulative execution time)} \qquad \text{(Def. 12)}$$

$$h(t_j) = \frac{-\ln(1-P_j)}{t_j} \qquad \text{(Eq. 11)}$$

Then, $$R(n) = \exp\left[-\sum_{i=1}^{n} \Delta t_j \; h(t_j)\right] \qquad \text{(Eq. 12)}$$

Equation 13 computes the conditional probability of failure during the interval $(t_j, t_j + \Delta t_j)$ given no failure prior to t_j as:

$$[\, h(t_j)] \times [\, \Delta t_j] \qquad \text{(Eq. 13)}$$

In summation, we have found no major flaws with this model. Although probability of failure and reliability are addressed, predicted times between failure or number of errors cannot be determined. Thus, for our particular problem, this model does not appear to be appropriate in that many aspects associated with it are concerned with problems that we are not currently addressing.

1.5.7 Pragmatic Software Reliability Prediction

Wall and Ferguson [135] initiate their discussion of software reliability by stressing the importance of identifying model parameters and using available software error data to estimate the current reliability as well as reliability at some future point in time. Then it would be possible to identify particular programs which are not performing up to specified standards.

They also claim that the analytical models that have been proposed for the error occurrence rate are eloquent yet impractical. These models, in their opinion, are not very useful to the system designer or program manager at the present state-of-the-art of software reliability for two reasons. First, "some of the relationships contain a large number of

parameters which must be empirically determined". Our search of the literature has produced over twenty models, none of which contain a "large" number of parameters. In fact, many of the models contain two parameters, which may be estimated fairly easily by utilizing a computer program. They also cite the limited amount of failure data available, which is a valid and accurate statement. Second, they claim, these analytic relationships do not seem to fit the total range of data very well.

Wall and Ferguson suggest that there is an algebraic relationship between the number of failures and the maturity of the software. Specifically, they give the equation:

$$C = C_o (M/M_o)^{\alpha} \qquad \text{(Eq. 1)}$$

where C = cumulative number of software failures experienced by a "set" of software. (Def. 1)

C_o = a constant (Def. 2)

α = a constant (Def. 3)

M_o = a scaling constant (Def. 4)

M = the "maturity" of the software (Def. 5)

The constants, α and C_o, must be determined empirically according to the authors, and they make a general statement that α lies between 0.3 and 0.7 for a wide range of program types. The range of α is extremely large and does not appear to be as useful as it may have been intended to be. Nevertheless, if equation 1 is differentiated with respect to time, we obtain an expression for the failure rate.

$$R = \alpha C_o \frac{d(M/M_o)}{dt} \left(\frac{M}{M_o}\right)^{\alpha-1} \qquad \text{(Eq. 2)}$$

Equation 2 may be rewritten as:

$$R = R_o \left(\frac{M}{M_o}\right)^{\alpha-1} \qquad \text{(Eq. 3)}$$

where R_o = a constant (Def. 6)

The authors do present plots of data which do seem to support their contention that there is reasonable correlation between the failure data and these functions. They do not, however, indicate what values they use for the constants in each data set, nor do they indicate how they might be estimated. In their paper, they do state, "a value, or even a range of values, for C_o and R_o is considerably more difficult to obtain because of the wide range of units for M,C, and R used to collect and report failure data". It is conjectured that they used a regression technique to obtain the best fitting curve in each case. As was stated, α appears to lie between 0.3 and 0.7, but no such range was able to be determined for C_o and R_o. This problem of estimation casts doubt on the usefulness of the model. We anticipate that a linear regression technique (using the logarithms of the proposed functions) may be an adequate means of estimating these constants for a particular model. However, due to some of the limitations and potential problems we have chosen not to actively pursue this method at this time.

1.5.8 *Regression Models*

Various regression models have been proposed which attempt to identify those software characteristics which influence software reliability [65,106]. These program characteristics include the number of statements in the program, the number of branches, the number of I/O instructions and some measure of complexity. Characteristics of this type reflect the nature of the program. Other characteristics which reflect the ability of the programmer include his years of academic experience, his years of operational experience, and a supervisor's rating. Of course, identifying such characteristics will not enable us to predict a failure rate or reliability directly at any point in time, but they do allow us to identify those principles and procedures which lead to the efficient development of reliable software. Further, if we can develop a viable prediction equation, we will be able to estimate the number

of errors we would expect to find based on the nature of the program and the programmer's ability. We could then use this prediction to refine our estimate of the number of errors that is derived using one of the analytic models.

Lipow and Thayer [65] present a "non-standard method of linear regression analysis" which evaluates predictors for the numbers of software errors. There are some serious questions which must be resolved concerning the validity of their method, which they themselves admit to be "very non-standard". First, they constrain the regression coefficients of the predictor (independent) variables to be non-negative. The rationale for the non-negativity constraints is so that the predictor variables do not have a negative impact on the number of software errors. Second, they assumed a zero-intercept for all regression equations. This second method is particularly questioned since a non-zero intercept would have little or no meaning for their range of data and tends to yield higher, but non-comparable, correlation coefficients than does the non-zero intercept model. In the text of their paper and on the corresponding graphs, certain points were designated statistical "outliers" and eliminated from their analysis. No valid reasoning was presented as to why the "outliers" were excluded, however, one hypothesizes that "better results" could be obtained by ignoring such points.

Schick and Wolverton [106] present a classical least squares regression analysis where a best fit may be determined. Scatter plots and averaging techniques were also used to determine possible relationships in the data. Furthermore, a goodness of fit technique was utilized which is described below:

$$r^2 = 1 - \left[1 - \frac{\sigma^2_{est.}}{\sigma^2_{observed}}\right]\left[\frac{n-1}{n-T}\right] \qquad \text{(Eq. 1)}$$

where r^2 = index of determination (Def. 1)

$\sigma^2_{est.}$ = variance of the values of the dependent variable predicted by the regression equation (Def. 2)

$\sigma^2_{observed}$ = variance of the actual values of the dependent variable (Def. 3)

n = total number of data points (Def. 4)

T = the number of terms in the regression equation (Def. 5)

If, in equation 1, r^2 approaches 1.0, then it may be said that the estimating equation does account for the observed variations in the dependent variable. Conversely, as r^2 approaches 0, the opposite would be true.

The results obtained by Schick and Wolverton show that the only significant predictor of the error content of a program and the cost of developing the routine is the size, or number of statements, in the routine. The effect of all other independent variables on the number of errors and the cost of the routine seems to be negligible although a large portion of the variation in the data remains unexplained. In particular, programmer experience seems to have little effect on errors or costs. They do caution, however, that only the most intuitive relationships were tested.

The two regression techniques presented sharply contrasted one another. Lipow and Thayer had some very questionable steps in their approach and analysis. Schick and Wolverton utilized a much more common linear regression analysis. However, it does not appear that the relationship between program characteristics or programmers ability and the reliability of the software will be of any great value in predicting the number of errors in a program.

1.6 ANALYTICAL MODELS

Thayer, Craig, and Frey [128] define a "software reliability model" as a mathematical model constructed for the purpose of assessing the reliability of software from specified parameters which are either assumed known or are estimated from observable data. We shall refer to models of

this type as "analytical models" since they describe mathematical relationships between the parameters and reliability. That is, reliability (or equivalently, failure rate) is depicted as a function of these parameters. After reviewing other reliability techniques (Sections 1.2 and 1.3) and other types of models (Sections 1.4 and 1.5) we feel that the analytical models offer the most realistic and accurate way of determining and predicting software reliability.

The first analytical software reliability models to our knowledge were developed in 1971, and we have found a total of fifteen developed since then. We have tried to carefully study, apply, and evaluate all of them. In this section we present all fifteen models and include a summary of each model, a discussion of the assumptions, an explanation of how to use the model, and finally a preliminary evaluation of each model relative to the intended use of the software (i.e. in a digital flight control system). In this preliminary evaluation eleven of the models for various reasons are deemed inappropriate for use in our efforts. The remaining four (Jelinski-Moranda, extended Jelinski-Moranda, Schneidewind, and Geometric) models show some definite promise and will be studied and evaluated in greater detail in Section 1.7.

1.6.1 Jelinski-Moranda Model

One of the earlier software error models was proposed by Jelinski and Moranda and may be shown to be equivalent to the Shooman exponential model [56], [79]. This so called "de-eutrophication" model makes the same basic assumptions as does the Shooman, and the two are nearly identical, with the exception that Jelinski and Moranda do not explicitly include the number of machine language instructions, although it is assumed that the number remains relatively constant. The parameters associated with the Jelinski-Moranda model, N and $\emptyset$, are considered fixed for the individual program and

are estimated via maximum likelihood (of course these estimates may change as errors are discovered). Jelinski and Moranda define the following notation for their model:

N = the total number of initial errors in the program (Def. 1)

$\emptyset$ = a proportionality constant (Def. 2)

X_i = the length of the i^{th} debugging interval (the time between detection of the $(i-1)^{st}$ and i^{th} errors) (Def. 3)

i = the number of errors discovered (equals the number of intervals) (Def. 4)

Now, the hazard function during the i^{th} debugging interval (after the detection of the $(i-1)^{st}$ error, but before the detection of the i^{th} error is

$$Z(X_i) = \emptyset\,[N-(i-1)] \qquad \text{(Eq. 1)}$$

Thus, in this manner, the hazard function $Z(X_i)$ is defined piecewise over time (see Figure 5).

Forman [38] examines this particular model in detail and, assuming that the model parameters are known, derives expressions for the number of bugs to be removed before the software is deemed acceptable. The distribution of the time to successfully debug the system is given, and bounds are also given for this distribution. When the model parameters are not known Forman presents three methods for their estimation; maximum likelihood, least squares and pseudo Bayesian.

Furthermore, Forman also attempts to expand the de-eutrophication model to incorporate the rate of error generation during the debugging process. He reports that good parameter estimates are difficult to attain unless the software is completely or nearly completely, debugged. As such, he concludes, this model does not appear to be applicable during the entire debugging process, but "can be used to answer the question of whether or not all bugs have been removed when it is believed debugging has been completed".

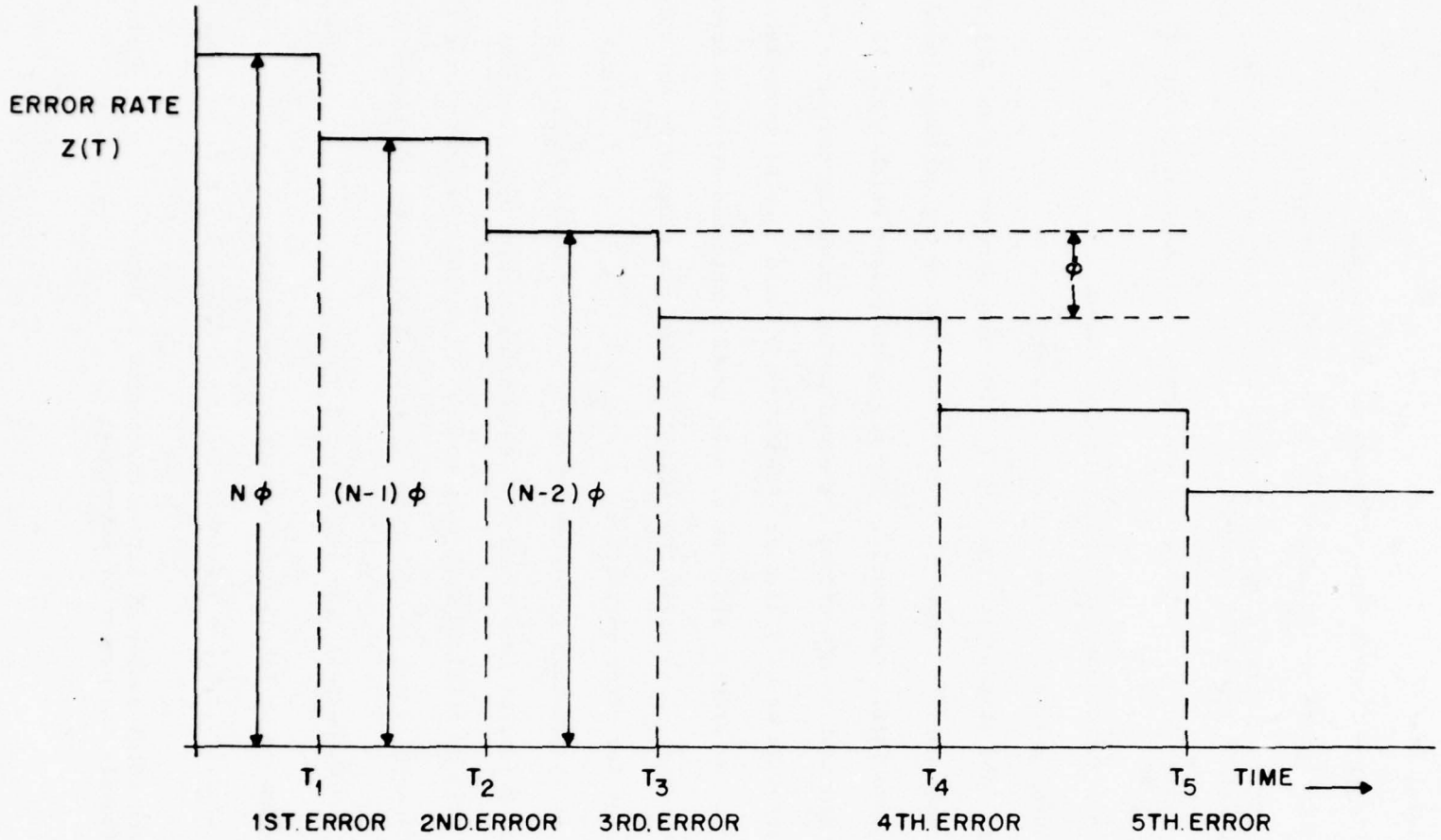


FIG. 5 – JELINSKI-MORANDA DE-EUTROPHICATION MODEL

FAILURE RATE IS PROPORTIONAL TO THE NUMBER OF ERRORS REMAINING.

The basic assumptions corresponding to the Jelinski-Moranda de-eutrophication model are:

(1) There is a fixed number of errors in the program.

(2) No new errors are added during the debugging process.

(3) The failure rate is proportional to the current error content (number of remaining errors).

(4) It is implicitly assumed that each error has an equal chance of being detected.

(5) Each error discovered is immediately removed.

(6) The failure rate between errors is constant.

Myers [86] states "It (the model) makes many assumptions and all of them are questionable". However, we feel that most of the aforementioned assumptions are fairly reasonable. The only assumptions which might be questioned are that no new errors are added during debugging and secondly, that each error is equally likely. Assumption 5 could also be interpreted as being that the program will not be used until a detected error is corrected. Finally, this model also makes the assumption that the program is not being altered except for error correction.

Now it is possible to proceed with the estimation of the model parameters, N and ϕ (definitions 1 and 2). As previously mentioned, maximum likelihood will be utilized with this model. In definition 3 we stated that X_i was the length of the i^{th} debugging interval and its density may be given as

$$P(X_i) = \phi\,[N-(i-1)]\,\exp\{-\phi[N-(i-1)]\,X_i\} \qquad \text{(Eq. 2)}$$

Now the corresponding likelihood function may be written as

$$L(X_1,X_2,\ldots\ldots,X_n) = \prod_{i=1}^{n} \phi\,[N-(i-1)]\,\exp\{-\phi[N-(i-1)]\,X_i\} \qquad \text{(Eq. 3)}$$

where n = the total number of errors discovered to date (equals the number of intervals). (Def. 5)

Taking the logarithm of the likelihood function yields:

$$\log_e L = \sum_{i=1}^{n} \log \{[N-(i-1)]\ \phi\} - \sum_{i=1}^{n} (N-(i-1))\ \phi\ X_i \qquad \text{(Eq. 4)}$$

or

$$\log_e L = \sum_{i=1}^{n} \log \{N-(i-1)\} + n \log \phi - \sum_{i=1}^{n} (N-(i-1)\ \phi\ X_i \qquad \text{(Eq. 5)}$$

Taking partial derivatives with respect to N and ϕ and then setting the resultant equations equal to zero gives:

$$\frac{\partial \log L}{\partial N} = \sum_{i=1}^{n} \frac{1}{N-(i-1)} - \sum_{i=1}^{n} \phi\ X_i = 0 \qquad \text{(Eq. 6)}$$

$$\frac{\partial \log L}{\partial \phi} = \frac{n}{\phi} - \sum_{i=1}^{n} (N-(i-1))\ X_i = 0 \qquad \text{(Eq. 7)}$$

Next, by letting $\Sigma X_i = T$ we can solve equation 7 for ϕ and obtain

$$\phi = \frac{n}{NT - \sum_{i=1}^{n} (i-1)\ X_i} \qquad \text{(Eq. 8)}$$

where T = the cumulative debugging time (total number of intervals) $= \Sigma X_i$ (Def. 6)

Then, equation 6 becomes

$$\sum_{i=1}^{n} \frac{1}{N-(i-1)} = \frac{n}{N - \frac{1}{T} \sum_{i=1}^{n} (i-1)\ X_i} \qquad \text{(Eq. 9)}$$

Since ϕ is no longer present in equation 9 solving this equation becomes the next step. The two data derived statistics are $T = \Sigma X_i$ and $\Sigma(i-1)X_i$ and knowing these we can expand equation 9 and then solve for N

$$\frac{1}{N} + \frac{1}{N-1} + \cdots\cdots\cdots + \frac{1}{N-(n-1)} = \frac{n}{N - \frac{1}{T} \sum_{i=1}^{n} (i-1)\ X_i} \qquad \text{(Eq. 10)}$$

The estimate for N, called $\hat{N}$, can be utilized to obtain an estimate for ϕ.

$$\hat{\phi} = \frac{n}{\hat{N}T - \sum_{i=1}^{n} (i-1)\ X_i} \qquad \text{(Eq. 11)}$$

rewriting the right hand side of equation 10 yields

$$\frac{1}{N} + \frac{1}{N-1} + \ldots\ldots\ldots + \frac{1}{N-(n-1)} = \frac{n}{N-P} \qquad \text{(Eq. 12)}$$

$$\text{where } P = \frac{\sum_{i=1}^{n} (i-1)\, X_i}{\sum_{i=1}^{n} X_i} \qquad \text{(Def. 7)}$$

Then multiplying both sides of equation 12 by (N-P) gives

$$\frac{N-P}{N} + \frac{N-P}{N-1} + \ldots\ldots\ldots\ldots + \frac{N-P}{N-(n-1)} = n \qquad \text{(Eq. 13)}$$

$$\left[\frac{N}{N} - \frac{P}{N}\right] + \left[\frac{N-1}{N-1} - \frac{P-1}{N-1}\right] + \ldots\ldots\ldots + \left[\frac{N-(n-1)}{N-(n-1)} - \frac{P-(n-1)}{N-(n-1)}\right] = n \qquad \text{(Eq. 14)}$$

Simplifying equation 14 results in

$$1 - \frac{P}{N} + 1 - \frac{P-1}{N-1} + \ldots\ldots\ldots + 1 - \frac{P-(n-1)}{N-(n-1)} = n \qquad \text{(Eq. 15)}$$

The above then reduces to

$$\frac{P}{N} + \frac{P-1}{N-1} + \ldots\ldots\ldots + \frac{P-(n-1)}{N-(n-1)} = 0 \qquad \text{(Eq. 16)}$$

The parameters of the Jelinski-Moranda model can be estimated using equations 9-12. Essentially, equations 9, 10 and 12 are equal with different notation (substituted) and are used to find $\hat{N}$. Equation 11 provides an estimate for $\emptyset$.

The only data required for the de-eutrophication model is the sequence of times between errors (i.e. X_1, X_2, X_3, $\ldots\ldots\ldots X_n$).

This model has been selected for further study in section 1.7.

1.6.2 Basic Shooman Model

One of the earliest software reliability models was proposed by Dr. Martin L. Shooman [116-118] of the Polytechnic Institute of New York. In 1971 Shooman developed this model which relates the probability of encountering a software error to the number of errors remaining in the software,

the total number of machine language instructions, and in his earlier papers, the instruction processing rate. One major objective was to examine error behavior during debugging which posseses some generality for both small and large programs, and then relate this behavior back to the probability of encountering an error.

The Shooman model may be applied in two different ways and a brief discussion is in order. First, this technique may be utilized through a two point estimation process. This approach is susceptible to varied results (predictions) depending on the pair of estimation points chosen. Further development of this particular method, including its application to software error data, will appear later in this section. The second alternative with Shooman's model is a full application of the model, and this approach will be examined in considerable detail following a presentation of some of the basic model concepts.

Initially, we will establish the following assumptions that Shooman specifies regarding his exponential model.

(1) There is a fixed number of errors in a program.

(2) No new errors are added during debugging.

(3) The error detection rate (failure rate) is proportional to the number of remaining errors.

(4) The number of machine language instructions is constant.

(5) Operational software errors occur due to the occasional traversing of a portion of the program in which a hidden software bug is lurking.

(6) Implicitly assumes that each error has an equal chance of being detected.

(7) Implicitly assumes that a relationship between operational time and debugging time can be determined.

It should be noted that Shooman based his model on some simplifying assumptions. First, let us evaluate the impact of the assumptions that there is a fixed number of errors in the program and that no new errors are generated during the debugging process. Stating that no new errors are introduced during debugging implies that the correction procedure is perfect and is not considered to be a very realistic assumption. Together, the two aforementioned assumptions infer that eventually we will reach the point where software is perfectly reliable -- a conclusion most practitioners will be unwilling to make no matter how good the program appeared (for a program of any reasonable size).

It has been further assumed that the error detection rate is proportional to the number of errors remaining in the program. This suggests that the failure rate is constant between the detection of consecutive errors and declines as an error is detected (see Figure 5). Implicit in this are the assumptions that software errors are independent of each other and that each error has an equal chance of being detected at a given point in time. Hence, these assumptions result in an exponential distribution for the times between error occurrences. Thus, errors occur according to a Poisson process whose parameter declines whenever an error is detected.

Although these assumptions may be questioned they do appear to be reasonable for approximate error behavior in Phases II, III and IV (see Figure 1). These phases were discussed in Section 1.1.3 (The Software Development Process). In fact, behavior in the latter two stages may fit these assumptions quite well. Nevertheless, these particular assumptions may be required in order to develop a workable model. Intuitively, however, we can appreciate the possibility that an error may be "hidden" by another error which has not been detected. Furthermore, it seems plausible that some errors will be more subtle than others; hence some errors would be less

likely to be detected. Overall, however, the equally likely assumption seems to have merit.

Dr. Shooman also assumed that the number of instructions in the program remains unchanged, which reflects the reasonable assumption that we are dealing with a relatively "mature" piece of software (one that is basically unmodified except for error corrections). Finally, he also assumes that the relationship between operational time and debugging time or effort is known.

Similar to many of the analytical models, the Shooman model can be characterized by an instantaneous error detection rate or hazard function Z(t). As previously mentioned, this model assumes that the hazard function is proportional to the number of errors remaining in the program. Specifically, the Shooman hazard function is given by:

$$Z(t) = C[\varepsilon_r(\tau)] \qquad \text{(Eq. 1)}$$

$$Z(t) = C\{[E_T/I_T] - \varepsilon_c(\tau)\} \qquad \text{(Eq. 2)}$$

$$Z(t) = C\{[E_T/I_T] - [E_c(\tau)/I_T]\} \qquad \text{(Eq. 3)}$$

$$Z(t) = \{[C/I_T]\ [E_T - E_c(\tau)]\} \qquad \text{(Eq. 4)}$$

where t = operating time of the system measure from its initial activation (Def. 1)

τ = the amount of debugging time since the start of system integration (Def. 2)

C = a proportionality constant (Def. 3)

$\varepsilon_r(\tau)$ = normalized number of errors remaining at time τ $(E_r(\tau)/I_T = (E_T/I_T - \varepsilon_c(\tau))$ (Def. 4)

E_T = the total number of errors at time $\tau = 0$ (Def. 5)

I_T = the total number of machine language instructions (Def. 6)

$E_c(\tau)$ = the cumulative number of errors corrected (detected) at time τ (Def. 7)

$\varepsilon_c(\tau)$ = normalized number of errors corrected at time $\tau = (E_c(\tau)/I_T)$ (Def. 8)

$E_T - E_c(\tau)$ = the number of errors remaining at time τ (Def. 9)

Prior to discussing the method for estimating C and E_T it should prove helpful to discuss two final assumptions. First, it is tacitly assumed that each error has the same severity (i.e. all errors are catastrophic). This assumption could be easily removed by categorizing the errors by severity and thus estimating a new E_T and C for each of the categories. Secondly, it was assumed that the software errors were corrected immediately since the failure rate changes whenever an error is detected. This could be handled by either not using the program until the error is corrected or simply not counting it again (which is a questionable procedure practically, since it does assume that the error will be corrected and gives a distorted view of the true error rate).

Let us now examine the testing programs for estimating the two unknowns, C and E_T. Initially, the software failure rate is defined as:

$$\lambda_{s_i} = X_{s_i} / H_i \qquad \text{(Eq. 5)}$$

where X_{s_i} = software failures found during H_i (Def. 10)

H_i = "locally" cumulative number of units of time <u>after</u> debugging time τ_i. (Def. 11)

Then,

$$MTTF_{s_i} = H_i / X_{s_i} \qquad \text{(Eq. 6)}$$

In [117] Shooman gives the following reliability and mean time to failure equations:

$$R(t,\tau) = \exp\left[-C\left(\frac{E_T}{I_T} - \varepsilon_c(\tau)\right) t\right] \qquad \text{(Eq. 7)}$$

$$MTTF(\tau) = \frac{1}{C\left[\frac{E_T}{I_T} - \varepsilon_c(\tau)\right]} \qquad \text{(Eq. 8)}$$

where τ = the debugging time (Def. 12)

t = operational time from initial activation (Def. 13)

The unknowns (C and E_T) can be evaluated by running a functional test after two different debugging times, τ_1 and τ_2, chosen so that $\tau_1 < \tau_2$ and $\varepsilon_c(\tau_1) < \varepsilon_c(\tau_2)$. Then, by equating the mean time to failure equations (equations 6 and 8) at times τ_1 and τ_2, we obtain equations 9 and 10.

$$\frac{H_1}{X_{s_1}} = \frac{1}{C[\frac{E_T}{I_T} - \varepsilon_c(\tau_1)]} \qquad \text{(Eq. 9)}$$

$$\frac{H_2}{X_{s_2}} = \frac{1}{C[\frac{E_T}{I_T} - \varepsilon_c(\tau_2)]} \qquad \text{(Eq. 10)}$$

Now it is possible to estimate E_T by taking the ratio of equations 9 and 10 and then applying equation 5. The resulting estimate is:

$$\hat{E}_T = \frac{I_T\left[(\lambda_{s_2} / \lambda_{s_1})\, \varepsilon_c(\tau_1) - \varepsilon_c(\tau_2)\right]}{(\lambda_{s_2} / \lambda_{s_1}) - 1} \qquad \text{(Eq. 11)}$$

Next, the proportionality constant can be estimated by:

$$\hat{C} = \frac{\lambda_{s_1}}{[\frac{E_T}{I_T} - \varepsilon_c(\tau_1)]} \qquad \text{(Eq. 12)}$$

Finally, the times between failures constitute the data required for the Shooman model. Let us now examine the two point estimation process when applied to actual software data. A few sample calculations will be given to demonstrate the mechanics of this particular approach. The data on the following page was obtained from reference [134].

Date	Errors	Cumulative Number of Errors	CPU time	Cumulative CPU time	Errors Per Unit of CPU time
1/12	8	8	0.50	0.50	16.00
1/15	7	15	0.60	1.10	11.67
1/16	1	16	0.65	1.75	1.54
1/17	8	24	1.90	3.65	4.21
1/18	16	40	1.59	5.24	10.06
1/19	18	58	8.83	14.07	2.04
1/22	13	71	9.94	24.01	1.31
1/23	8	79	7.25	31.26	1.10
1/24	9	88	8.34	39.60	1.08
1/25	2	90	3.86	43.46	0.52
1/26	6	96	13.11	56.57	0.46
1/27	3	99	34.15	90.72	0.09
1/29	3	102	82.70	173.42	0.036
1/30	2	104	1.10	174.52	1.18
1/31	3	107	51.59	226.11	0.06

Fig. 6 F-11D Program Error Data

As can be seen in column 6 of Figure 6, the error behavior is somewhat erratic through day 1/18. Thus, day 1/18 will be analyzed along with day 1/19 (chosen arbitrarily). From equation 5 we can calculate the following:

$$\lambda_{s_1} = 18/8.83 = 2.038505$$

$$\lambda_{s_2} = 13/9.94 = 1.307847$$

For simplicity, let us set $I_T = 1$. Moranda, in [80], states "it is clear that the number of instructions I_T is only a nuisance since it is taken out by the 'normalization' of $\varepsilon_c(\tau_1)$ and $\varepsilon_c(\tau_2)$, which is required to produce numerical values". Then, by definition 8, we have

$$\varepsilon_c(\tau_1) = 40$$

$$\varepsilon_c(\tau_2) = 58$$

Hence, from equation 11, we can estimate E_T.

$$\hat{E}_T = \frac{[25.66286568 - 58]}{[0.641572 - 1]} = 90.219$$

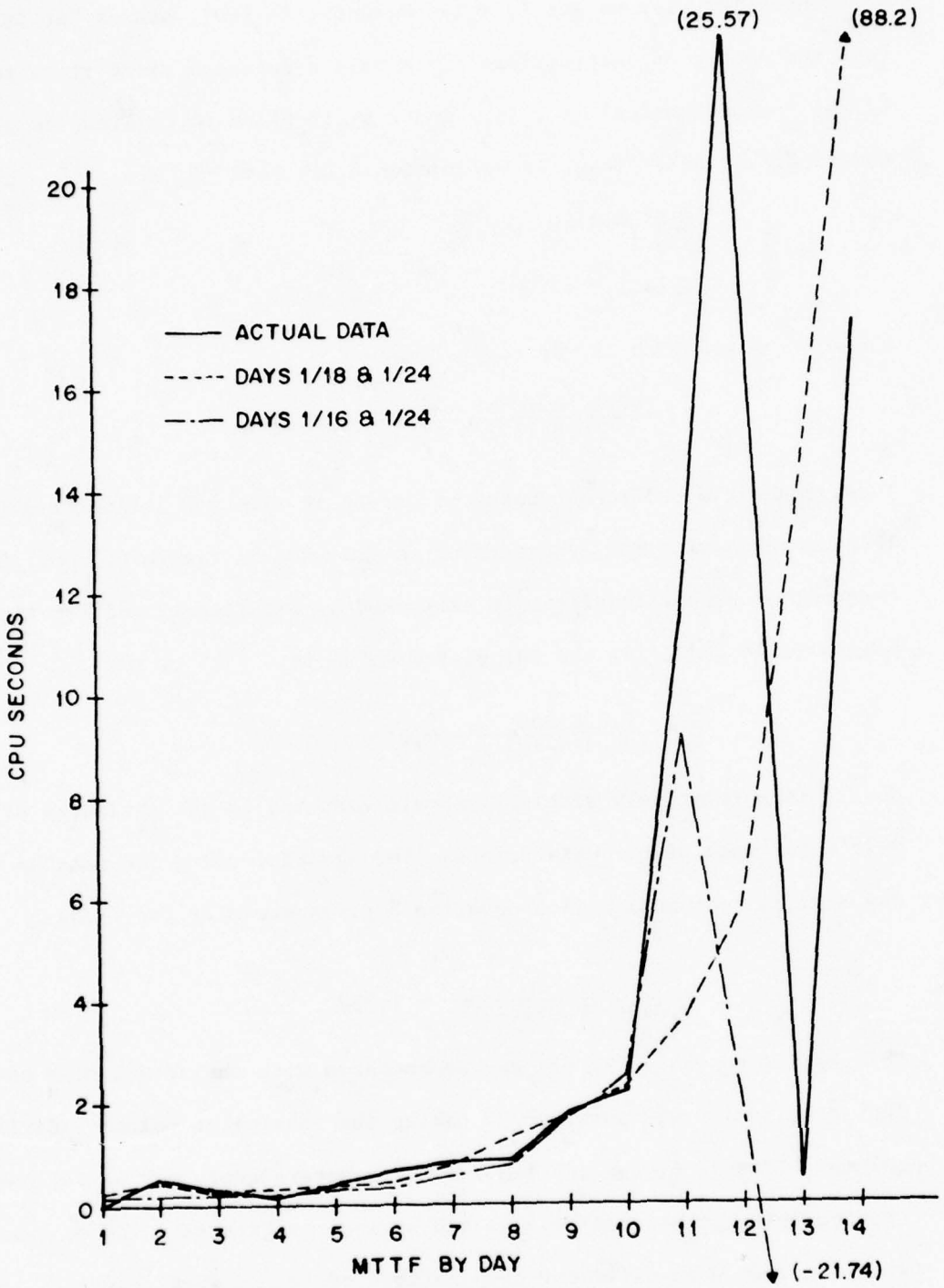
Thus, the estimated total number of errors at time τ=0 is approximately 90. This compares with 107 errors found as reported in Figure 6. Now, C, the constant of proportionality, is estimated by equation 12 and for these two points (1/18 and 1/19) the calculated value is:

$$\hat{C} = \frac{2.0385050}{[90.219-40]} = 0.0406$$

Now it is a relatively simple algebraic exercise to get estimates of the MTTF's for each of our data points. For example, using the data up through day 1/22 in conjunction with equation 8 gives the MTTF for 1/23.

$$\frac{1}{0.0406\ (90.219-71)} = 1.282$$

This estimated value (1.282) may be compared with the actual MTTF of 0.90625. The actual value was obtained by taking the inverse of column 2 divided by column 4 (Figure 6) for day 1/23. This same procedure may be followed for the remaining points (days). As can be seen in Figures 7 and 8, varied results were obtained when different combinations of points were selected. Some of the curves were better predictors than others, however, even with this small set of data points (15 points) there are 105 $\binom{15}{2}$ possible sets of points to choose



SHOOMAN'S EXPONENTIAL MODEL
(2 POINT ESTIMATE)
WAGONER'S DATA- F-11D PROGRAM

FIG. 7

subsequent date	actual MTTF	1/18 & 1/19	1/18 & 1/22	1/18 & 1/23	1/18 & 1/24	1/16 & 1/24	1/22 & 1/25
1/15	0.086	0.2996	0.3539	0.3329	0.3276	0.2164	0.3393
1/16	0.650	0.3275	0.3769	0.3581	0.3533	0.2346	0.3657
1/17	0.2375	0.3319	0.3804	0.3620	0.3573	0.2375	0.3698
1/18	0.0994	0.3720	0.4112	0.3966	0.3929	0.2631	0.4063
1/19	0.4906	0.4906	0.4906	0.4906	0.4906	0.3356	0.5062
1/22	0.7646	0.7646	0.6267	0.6686	0.6810	0.4862	0.6997
1/23	0.906	1.282	0.7837	0.9062	0.9464	0.7195	0.9667
1/24	0.927	2.196	0.9267	1.160	1.245	1.021	1.263
1/25	1.930	11.101	1.166	1.693	1.930	1.930	1.930
1/26	2.185	112.366	1.237	1.886	2.199	2.406	2.185
1/27	11.383	-4.262	1.513	2.862	3.777	9.272	3.630
1/29	25.57	-2.806	1.703	3.863	5.892	-21.74	5.422
1/30	0.55	-2.091	1.948	5.939	13.389	-5.004	10.712
1/31	17.20	-1.788	2.154	9.254	88.188	-3.307	30.640
--	$\hat{E}_T$	90.22	107.58	122.87	104.36	98.15	103.47
--	$\hat{C}$	0.0406	0.0302	0.0246	0.0317	0.0512	0.0340

Fig. 8 Shooman Model (2pt. est.) Applied to F-11D Data

from. Nevertheless, we do not know if any of the curves is in fact the best predictor. Figure 8 shows that days 1/18 and 1/22 produce an E_T of 107.58 which is the best estimate for the pairs of points we analyzed. However, there are over 100 additional pairs which we did not calculate E_T's for. One possible approach to the above problem (not being able to consider all combinations of points) is to take a reasonable number of pairs and calculate a mean value from the individual estimates. Utilizing Figure 8, with an n of 6, we find E_T = 104.5.

The estimated MTTF's (Figure 8) show considerable variation with one problem quite evident. If the estimated E_T is less than the actual number of errors (107 in this example) negative MTTF's may occur as shown in columns 3 and 7. Of course, such values are meaningless in a reliability analysis.

At this point we shall look at the full Shooman model and show that it reduces to the previously discussed Jelinski-Moranda model. Jelinski and Moranda proposed a hazard function of the form

$$Z(X_i) = \emptyset\ [N - (i-1)] \qquad \text{(Eq. 13)}$$

where X_i = the length of the $i^{\underline{th}}$ debugging interval (the time between the $i^{\underline{th}}$ and $(i-1)^{\underline{st}}$ errors (Def. 14)

$\emptyset$ = a proportionality constant (Def. 15)

N = the total number of initial errors (Def. 16)

i = the number of errors discovered after i intervals (Def. 17)

Recalling the Shooman hazard function as being

$$Z(t) = C\ (\varepsilon_r\ (\tau)\ ) \qquad \text{(Eq. 14)}$$

where $$\varepsilon_r(\tau) = \frac{E_T}{I_T} - \varepsilon_c(\tau) \qquad \text{(Eq. 15)}$$

we can show that the two respective hazard functions (equations 13 and 14) are identical if

$$E_T = N \qquad \text{(Def. 18)}$$

$$\frac{C}{I_T} = \emptyset \qquad \text{(Def. 19)}$$

$$\varepsilon_c\ (\tau) = \frac{i-1}{I_T} \qquad \text{(Def. 20)}$$

In [117] Shooman states that definitions 18 and 19 are equal with the authors merely selecting different notation. Definition 20 would be true if $\varepsilon_c(\tau) = i/I_T$.

Utilizing previous definitions (Def. 1-7) associated with the Shooman model and making the required substitution will show that the Shooman exponential model does reduce to the Jelinski-Moranda model. Below is the step by step proof.

$$Z(t) = C(\varepsilon_r(\tau)) \qquad \text{(Eq. 16)}$$

Substituting equation 15 in for $\varepsilon_r(\tau)$ yields

$$Z(t) = C\left[\frac{E_T}{I_T} - \varepsilon_c(\tau)\right] \qquad \text{(Eq. 17)}$$

From definition 20 we have

$$Z(t) = C\left[\frac{E_T}{I_T} - \frac{(i-1)}{I_T}\right] \qquad \text{(Eq. 18)}$$

Bringing the I_T term outside the parenthesis results in

$$Z(t) = \frac{C}{I_T}\left[E_T - (i-1)\right] \qquad \text{(Eq. 19)}$$

Now we have previously defined $\phi = \frac{C}{I_T}$ and $N = E_T$ so that

$$Z(t) = \phi\,[N - (i-1)] \qquad \text{(Eq. 20)}$$

Thus, equation 20 shows that the two models (Shooman and Jelinski-Moranda) do have identical hazard functions. Thus, the outcome of applying either model should produce identical results.

The Shooman model has been adapted by Dickson et.al. [30] with a moderate degree of success. It also was the basis of a study and extension by Miyamoto [77] (to be discussed later) which attempts to include a non-zero probability of generating new errors.

1.6.3 Extended Jelinski-Moranda Model

As we have shown, both the Shooman exponential model and the Jelinski-Moranda model require a sequence of times between failures in order to estimate

their parameters. It may be the case, however, that such data is not available, and we may only have data on the number of errors which occurred in a time interval. If this is the case, then the models can easily be extended and adapted to this situation.

The original models assume a constant failure rate between consecutive errors. One extension (essentially due to Lipow) approximates that behavior by assuming that the failure rate is constant over a time interval. Within the time interval, failures occur according to a Poisson distribution with the constant failure rate as its parameter. Between time intervals, the failure rate declines proportionally to the cumulative number of errors detected in the previous intervals (Figure 9). The basic assumptions associated with the extended Jelinski-Moranda model are the same as those of the basic Jelinski-Moranda model. Thus, the reader may refer back to the section covering the basic model for the necessary assumptions. With the extended model it is also assumed that more than one error may occur in a given time debugging period.

Stated symbolically, if we let n_i be the cumulative number of errors found through the $i^{\underline{th}}$ time interval, then the hazard function during the $i^{\underline{th}}$ time interval is:

$$Z(t_i) = \emptyset\,[N - n_{i-1}] \qquad \text{(Eq. 1)}$$

where $\emptyset$ = a proportionality constant (Def. 1)

N = the total number of initial errors (Def. 2)

n_i = the cumulative number of errors found through the $i^{\underline{th}}$ time interval (Def. 3)

t_i = the $i^{\underline{th}}$ debugging interval (Def. 4)

Below is a description of the approach to be used in estimating the model parameters. We would begin with the density function and the corresponding likelihood function. After taking the logarithms and partial derivatives with respect to $\emptyset$ and N we are left with equations 2 and 3.

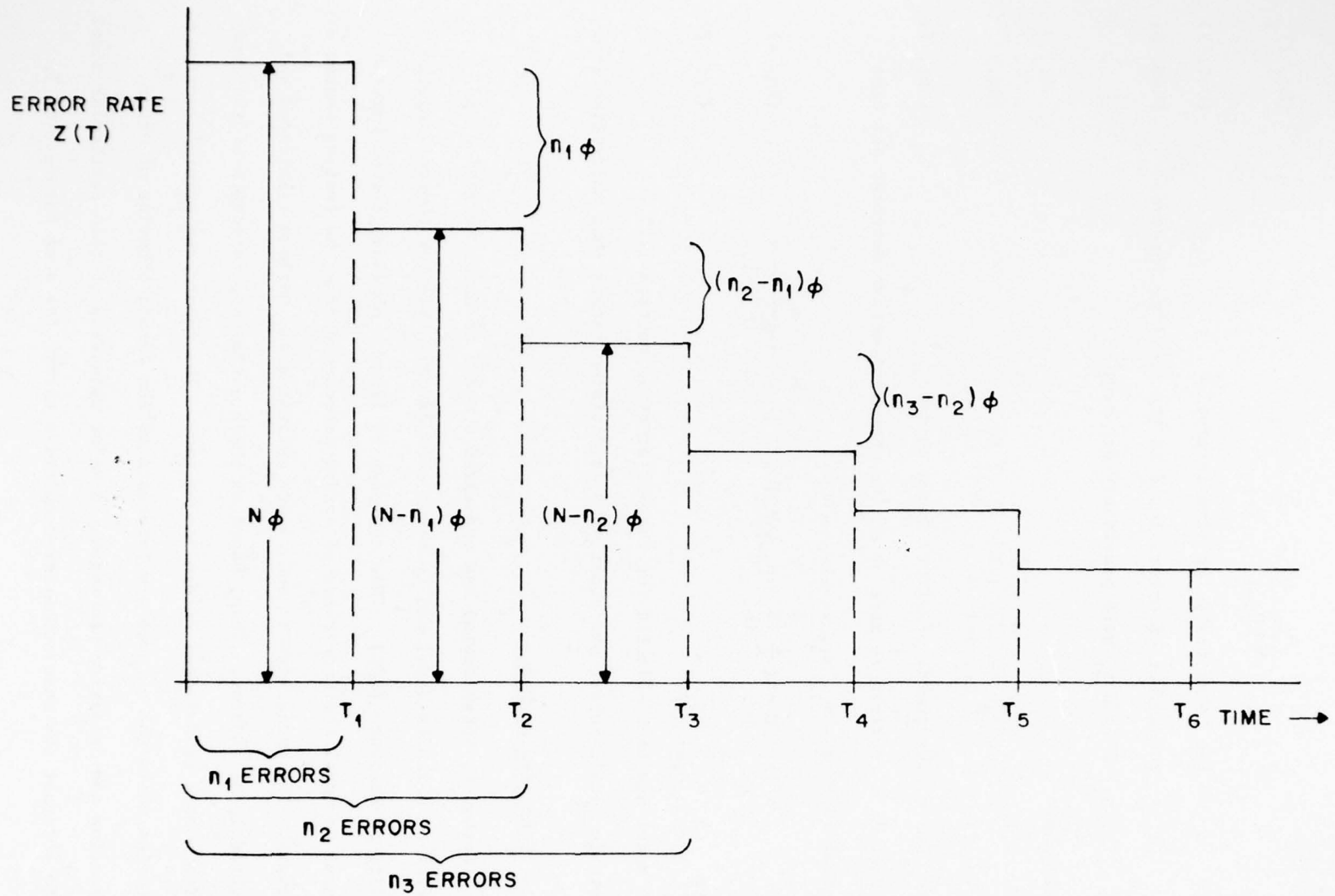


FIG. 9– EXTENDED JELINSKI-MORANDA MODEL

FAILURE RATE DURING AN INTERVAL IS PROPORTIONAL TO THE NUMBER OF REMAINING ERRORS AT THE BEGINNING OF THE INTERVAL.

$$\sum_{i=1}^{m} \frac{m_i}{N - n_{i-1}} - \sum_{i=1}^{m} \emptyset\, t_i = 0 \qquad \text{(Eq. 2)}$$

where m = the total number of time intervals (Def. 5)

m_i = the number of errors found in the $i^{\underline{th}}$ time interval (Def. 6)

N, n_i, $\emptyset$ and t_i were previously defined.

We also have

$$\frac{n}{\emptyset} - \sum_{i=1}^{m} (N - n_{i-1})\, t_i = 0 \qquad \text{(Eq. 3)}$$

where n = the number of errors found to date (Def. 7)

The test data required is made up of the number of errors detected per time interval (i.e. m_1, m_2, m_3, ..., m_m).

$$n = [N - (\frac{1}{A} \sum_{i=1}^{m} n_{i-1}\, t_i)] \quad [\sum_{i=1}^{m} \frac{m_i}{N - n_{i-1}}] \qquad \text{(Eq. 4)}$$

where $A = \sum_{i=1}^{m} t_i$ (Def. 8)

This model has been selected for further study in section 1.7.

Note: In [124], Sukert had equation 4 as division rather than multiplication.

1.6.4 Schneidewind Model

Norman F. Schneidewind has presented a model of error detection and correction which is developed to serve as a decision aid for controlling the quality of software [109]. This approach utilized a non-homogeneous Poisson process to model the occurrence of errors detected during the testing phases of software. The parameters (α and β) are estimated via maximum likelihood and weighted least squares. Then, based on these estimates, forecasts of the cumulative number of detected errors can be made. The inputs are error detection histories while the outputs are forecasts of the future behavior of error correction and detection processes. A major objective of this particular model is to forecast the mean number of cumulative errors for some future time T.

Schneidewind, in [109], makes the following point. "Unlike hardware which wears out or deteriorates with time, software should improve with time as more of the latent errors are detected and corrected. However, there are exceptions to this general characteristic. When an error is removed, it is possible that one or more errors will be introduced".

Then, in [138] Schick and Wolverton further state that "errors may reside undetected in software for many years until a particular set of input data causes a previously untraversed module path to be executed".

Following a listing of the assumptions, the various methods of this model will be analyzed.

(1) The number of errors which is detected during a time interval and the collection of error counts over a series of time intervals are modelled by a random variable and a stochastic process.

(2) Prior to the selection of a test plan, all errors are equally likely.

(3) The number of errors detected in each time interval is independent of the number detected in another time interval.

(4) Detected error counts in each interval have the same form (type of distribution) but have different means.

(5) The mean number of detected errors decreases from interval to interval.

(6) The rate of detection in an interval is proportional to the number of errors in that interval.

(7) Specifically, the detected error process is assumed to be a non-homogeneous Poisson process with an exponentially decreasing intensity function (failure rate).

(8) The error correction rate is proportional to the number of errors to be corrected.

The assumptions of this model seem to be straightforward, and we feel that they are also fairly realistic. Also, as with many of the other analytical models, the required test data consists of the sequence of number of errors in each interval.

Since the error process appears to change over time, recent error data would seem to be more useful than earlier data. Thus, we must carefully consider the following statements:

(1) to what extent should historical data be considered in forecasting, and

(2) how much of the historical time record to include when estimating α and β.

Based on these points, Schneidewind presents three approaches for this model which utilize the available data in different ways. First, with method 1, all error counts in intervals 1 through t are used,

where t = the last interval for which data is available (Def. 1)

Schneidewind further states that this particular method is appropriate if the changes in the error counts from intervals 1 to t are representative of the future ability to detect errors. In such a case, maximum likelihood can be applied to all error counts (X_i) for intervals 1 through t. Below is the approach to be used for method 1.

First we define,

β = a model constant (Def. 2)

α = a model constant (error detection rate at time 0) (Def. 3)

Now, to determine β, the following polynomial must be solved for y.

$$Ay^{t+1} - (A+1)y^t + (t-A)y + (A+1-t) = 0 \qquad y>1 \qquad \text{(Eq. 1)}$$

where $$A = \frac{\sum_{k=0}^{t-1} kX_{k+1}}{\sum_{k=1}^{t} X_k} \qquad \text{(Def. 4)}$$

X_k = the number of errors found in interval k (Def. 5)

t = previously defined (Def. 1)

$y = e^{\beta}$ (Def. 6)

Thus, $\beta = \ln y$ (Def. 7)

Now, α is determined by

$$\alpha = \frac{\left(\sum_{k=1}^{t} X_k\right)\beta}{1-e^{-\beta t}} \qquad \text{(Eq. 2)}$$

Upon obtaining estimates of α and β we can find the predicted number of errors for each interval i by

$$m_i = (\alpha/\beta)\,[\exp(-\beta(i-1)) - \exp(-\beta i)] \qquad \text{(Eq. 3)}$$

where m_i = the estimated number of errors in interval i (Def. 8)

Next, for method 2, a different approach is taken. Here, none of the error counts in intervals 1 through s-1 are used ($2 \leq s \leq t$) and all intervals from s to t are used.

s = an index with unit increment (Def. 9)

This type of an approach is appropriate if the most recent observations (intervals s through t) appear to be more representative of the future ability to detect errors. With this method, some criteria for selecting s is necessary. In such a case, α and β must be determined, again by maximum likelihood for all values of s from 2 through t. Once again we begin by solving a polynomial for y.

$$Ay^{C+1} - (A+1)y^{C} + (C-A)y + (A+1-C) = 0 \qquad y>1 \qquad \text{(Eq. 4)}$$

where $A = \dfrac{\sum_{k=0}^{t-s} kX_{s+k}}{\sum_{k=1}^{t} X_k}$ (Def. 10)

$$C = t-s+1 \qquad \text{(Def. 11)}$$

$$\beta = \ln y \qquad \text{(Def. 12)}$$

Then, $$\alpha = \frac{\left(\sum_{k=1}^{t} X_k\right)\beta}{1-e^{-\beta t}} \qquad \text{(Eq. 5)}$$

In effect, the results will give α_s and β_s for all s such that $2 \leq s \leq t$. Now, the best results are desired, and Schneidewind discusses a technique for

determining the optimal values. He suggests computing the sum of weighted squared deviations, SD_W, between m_i and X_i from intervals 1 to t (intervals are of equal length).

m_i = see definition 8

X_i = the actual number of errors in interval i (Def. 13)

The approach of using SD_W, which is also used in conjunction with method 3, will be discussed in more detail following a presentation of the third method.

With method 3, the cumulative error count from intervals 1 through s-1 is used and the individual error counts in intervals from s to t are also used. This particular method is appropriate if the individual error counts in intervals 1 to s-1 are not representative of the ability to predict the future but the cumulative count is, and the individual error counts from s to t are also representative.

Method 3, somewhat similar to method 2, requires first the estimation of α and β for all values of s from 2 through t. The following polynomial should then be solved for y.

$$Ay^{s+t} - (A+X_{s,t})y^{s+t-1} - (A+sX_{s-1}-X_{s-1})y^{t+1} + (A+X_{s,t}+sX_{s-1}-X_{s-1})y^{t} - (A-tX_t)y^{s} +$$

$$(A+X_{s,t}-tX_t)y^{s-1} + (A+sX_{s-1}-tX_t-X_{s-1})y - (A+sX_{s-1}+X_{s,t}-tX_t-X_{s-1}) = 0 \quad \text{(Eq. 6)}$$

where $A = \sum_{k=0}^{t-s} (s+K-1)X_{s,k}$ (Def. 14)

$$X_{s,t} = \sum_{k=s}^{t} X_k \quad \text{(Def. 15)}$$

$$X_{s-1} = \sum_{k=1}^{s-1} X_k \quad \text{(Def. 16)}$$

$$X_t = \sum_{k=1}^{t} X_k \quad \text{(Def. 17)}$$

$$\beta = \ln y \quad \text{(Def. 18)}$$

Now,

$$\alpha = \frac{\left(\sum_{k=1}^{t} X_k\right)\beta}{1 - e^{-\beta t}} \qquad \text{(Eq. 7)}$$

Let us now consider in more detail the weighted squared deviations criteria for selecting the optimal results associated with methods 2 and 3. The following formula can be utilized with both respective methods and is given as

$$SD_W = \sum_{k=1}^{t} \exp(\beta i)\ [(\alpha/\beta)[\exp(-\beta i)][\exp(\beta) - 1] - X_i]^2 \qquad \text{(Eq. 8)}$$

That is, you substitute in α_2 and β_2 to get SD_{W2}, α_3 and β_3 to get SD_{W3}, ... α_t and β_t to get SD_{Wt}. Then, you select the SD_W that is a minimum and record the associated positive values of α and β .

The above method is used on methods 2 and 3 only and compares the SD_W within methods. If we wish to compare between methods 1, 2 and 3, unweighted squared deviations are computed for each method, and the minimum value constitutes the optimal result. These unweighted squared deviations are calculated between forecasted errors and actual errors in intervals t+1 to T,

where T = some future time. (Def. 19)

Finally, Schneidewind also discusses the error detection rate and error correction rate in terms of α's, β's and i's. A more complete explanation of this material is covered in [109]. The Schneidewind method 1 and method 3 (s=2) are both equivalent to the Geometric-Poisson model. A proof of this has been included in section 1.6.6 which discusses the Geometric-Poisson model.

This model has been selected for further study and will be applied to data in order to better evaluate its utility in section 1.7.

1.6.5 Geometric Model

P. B. Moranda, a co-developer of the previously discussed de-eutrophication model, has also developed a geometric model which exhibits many similarities to the earlier model. However, certain variations are evident, among them being that this particular model does not assume a fixed, finite number of errors nor

does it assume that each error has the same likelihood of detection. On the contrary, the geometric model assumes that successive errors become harder to find (which is likely to occur in a relatively mature piece of software). Moranda contends that the assumption of a non-finite number of errors may be realistic [79]. "There are those who believe that there are not a finite number of errors in a large real time program; certainly this is so if there is an attempt to mirror in software all of the continuum of eventualities which occur in complex dynamic situations." This same paper further states, "the assumption that all errors have the same likelihood of detection is also an imperfect rendition of the real situation".

The following assumptions relate to the geometric model.

(1) Assumes that there are an infinite number of total errors.

(2) Errors do not have the same likelihood of detection.

(3) The failure rate between successive errors forms a geometric progression and is constant in the interval between errors.

(4) Each error discovered is immediately removed, thus decreasing the number of errors by one.

Essentially, this model is based on the assumption that the failure rate between successive errors forms a geometric progression. The error rate in an interval is proportional to the error rate before the detection of the last error (see Figure 10). It (the model) makes no assumptions about error generation, but allows for the possibility since the model does assume that there are always errors remaining (a very practical assumption for a programmer to make). If we assume, as Belady and Lehman did [9], that at each error correction, a number E of errors is extracted and an amount G is generated, both of which are fractions of the number of remaining errors at the time the error is corrected, then this geometric model is appropriate. In essence, this model does not make any assumptions about the mechanizations which cause errors; instead it models how the errors appear to behave.

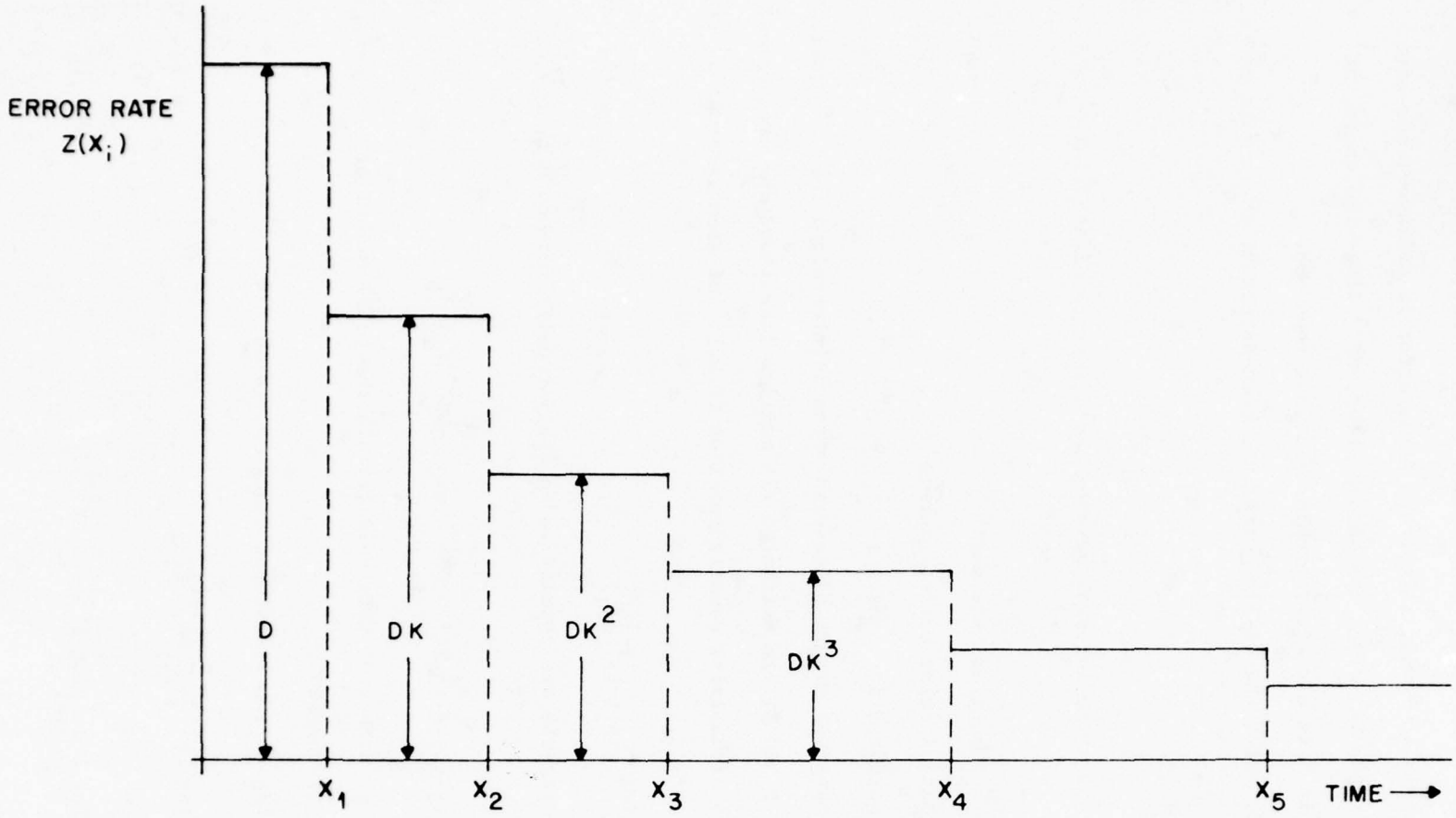


FIG 10 - GEOMETRIC DE - EUTROPHICATION MODEL

ERROR RATE DECLINES GEOMETRICALLY BETWEEN THE DETECTION OF SUCCESSIVE ERRORS, i.e., THE ERROR RATE IS PROPORTIONAL TO THE RATE BEFORE THE DETECTION OF THE LAST ERROR.

With the geometric model, the initial error detection rate is constant until the first error is found. After the first error is detected, the error detection rate becomes DK, after the second error, DK^2, after the third, DK^3 and so forth, where K is a positive number and less than one.

In general, after the $(i-1)^{\underline{st}}$ error has been detected, the hazard function is:

$$Z(X_i) = DK^{i-1} \qquad \text{(Eq. 1)}$$

which is constant over the interval between the detection of the $(i-1)^{\underline{st}}$ and $i^{\underline{th}}$ errors.

X_i = the $i^{\underline{th}}$ debugging interval (Def. 1)

D = the initial error detection rate (Def. 2)

K = a proportionality constant (Def. 3)

i = the number of errors discovered after i intervals (Def. 4)

These parameters, D and K, are estimated via maximum likelihood and are derived below. First, the probability density function (P.D.F.) of each interval X_i is given as:

$$p(X_i) = DK^{i-1} \exp\ (-DK^{i-1}X_i) \qquad \text{(Eq. 2)}$$

and because the intervals are assumed to be statistically independent, the likelihood function is:

$$L(X_1,X_2,X_3,.....,X_n) = \prod_{i=1}^{n} DK^{i-1} \exp\ (-DK^{i-1}X_i) \qquad \text{(Eq. 3)}$$

where n = the total number of intervals (equals the total number of errors discovered). (Def. 5)

Next, taking the log of the likelihood function yields

$$\log L = \sum_{i=1}^{n} \log DK^{i-1} - \sum_{i=1}^{n} DK^{i-1}X_i \qquad \text{(Eq. 4)}$$

or

$$\log L = n \log D + \sum_{i=1}^{n} \log K^{i-1} - \sum_{i=1}^{n} DK^{i-1}X_i \qquad \text{(Eq. 5)}$$

By taking the partial derivatives with respect to D and K we obtain

$$\frac{\partial \log L}{\partial D} = \frac{n}{D} - \sum_{i=1}^{n} K^{i-1} X_i = 0 \qquad \text{(Eq. 6)}$$

$$\frac{\partial \log L}{\partial K} = \frac{1}{K} \sum_{i=1}^{n} (i-1) - D \sum_{i=1}^{n} (i-1) K^{i-2} X_i = 0 \qquad \text{(Eq. 7)}$$

Solving equations 6 and 7 will give the following expression

$$\frac{\sum_{i=1}^{n} iK^i X_i}{\sum_{i=1}^{n} K^i X_i} = \frac{n+1}{2} \qquad \text{(Eq. 8)}$$

Now, if the above equation is solved for K, a solution for D may be found by

$$D = \frac{n}{\sum_{i=1}^{n} K^{i-1} X_i} \qquad \text{(Eq. 9)}$$

The test data necessary to apply this model is the sequence of times between errors (i.e. $X_1, X_2, X_3, \ldots\ldots, X_n$).

Finally, the mean time to failure (MTTF) and the reliability may be estimated by equations 10 and 11 respectively.

$$\text{MTTF} = \frac{1}{DK^n} \qquad \text{(Eq. 10)}$$

$$R(X_i) = \exp[-DK^n X_i] \qquad \text{(Eq. 11)}$$

This model has been selected for further study in section 1.7.

1.6.6 Geometric-Poisson Model

As we have seen before, software error data may also be classified as failures per interval, and the nature of this error detection process suggests the Poisson distribution as descriptive of the number of errors detected in a final time period. Moranda discusses this extension in detail in reference [79]. "Although there is, in reality, a continual purification which takes place within a given time period, expediency requires that the detection rate be assumed constant over time. If the purification is known to be only partially and, in the best case, insignificantly, accomplished during the first time period, then there

is merit to the assumption of a constant rate. This of course, will be the situation if the time period used is short relative to the total development time."

Under these conditions, the Poisson distribution with parameter λ can be used to describe the errors detected in the first time period. During the second time period, the average number of errors detected is assumed to be proportional to the average number detected in the first interval (λ), and so forth for additional time periods (Figure 11). Thus, the average number of errors for each successive time interval forms a geometric progression ($\lambda, \lambda K, \lambda K^2, \lambda K^3 \ldots\ldots$) and the hazard rate during the $i^{\underline{th}}$ time interval is:

$$Z(t_i) = \lambda K^{i-1} \qquad \text{(Eq. 1)}$$

where t_i = the $i^{\underline{th}}$ debugging interval (Def. 1)

λ = the initial detection rate (Def. 2)

K = a proportionality constant (Def. 3)

Below is a complete listing of the assumptions for the Geometric-Poisson model.

(1) There is a non-finite number of errors.

(2) Non-equal likelihood of error detection.

(3) During a fixed interval of time, the number of errors detected follows a Poisson distribution.

(4) During each of these periods of time (data collection intervals) the detection rate (parameter of the Poisson distribution) is constant.

(5) Data is available only at discrete intervals.

(6) The detection rate in successive time intervals forms a geometric progression.

(7) Each error discovered is immediately removed or no longer counted.

The test data necessary for utilization of this model is the sequence of number of errors detected per time interval.

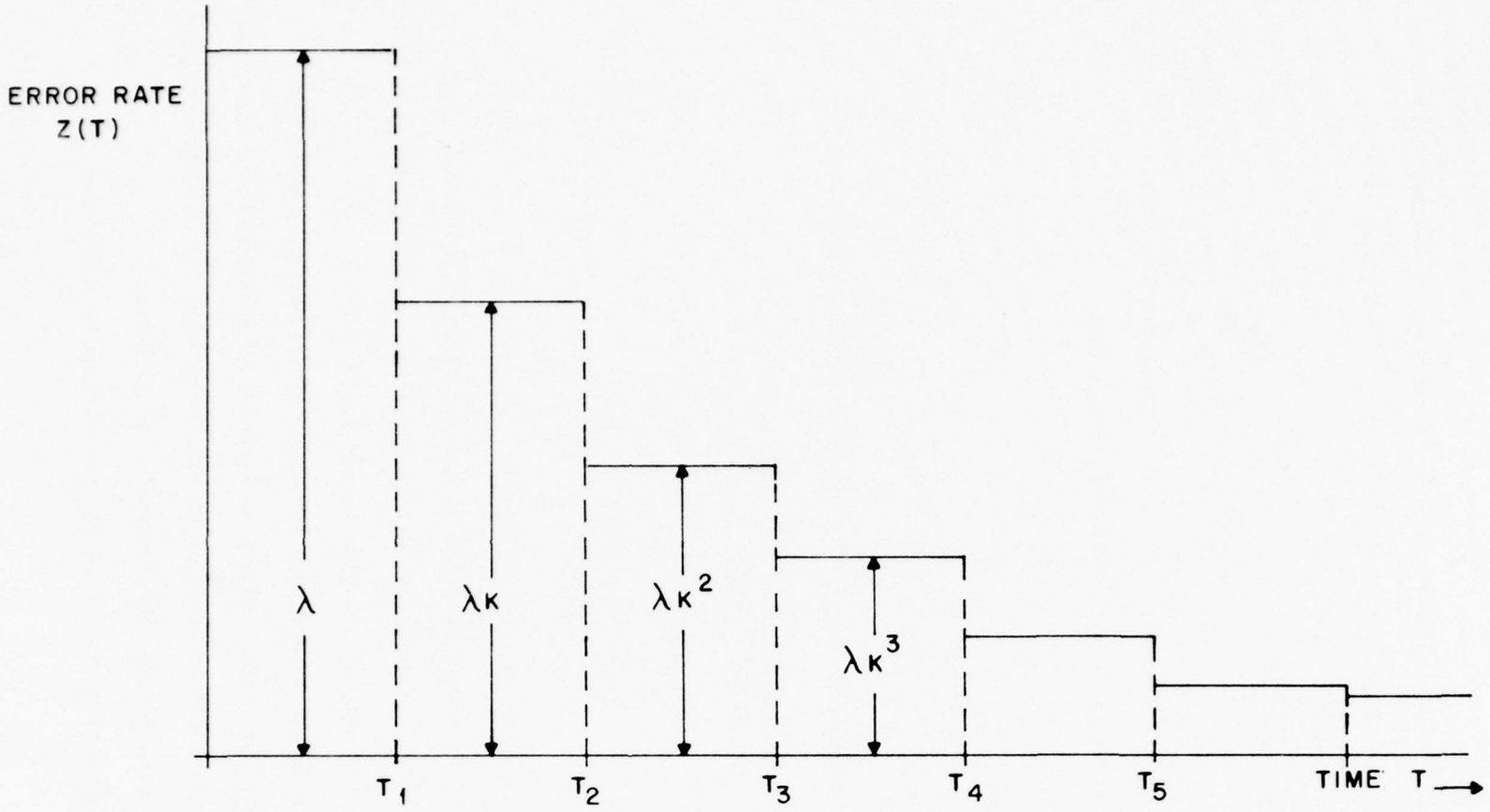


FIG. 11 - GEOMETRIC-POISSON MODEL

ERROR RATE IS PROPORTIONAL TO THE ERROR RATE IN THE PREVIOUS INTERVAL.

Following is a brief explanation of the derivation of the parameter estimators. The probability density function (P.D.F.) is given as:

$$P(n_i) = (\lambda K^{i-1})^{n_i} \exp(-\lambda K^{i-1}) \qquad \text{(Eq. 2)}$$

Then, the likelihood function is:

$$L(n_1, n_2, \ldots\ldots, n_m) = \prod_{i=1}^{m} (\lambda K^{i-1}) \exp(-\lambda K^{i-1}) \qquad \text{(Eq. 3)}$$

Taking the log of the likelihood function and then taking the partial derivatives with respect to λ and K yields the following equations:

$$1/\lambda \sum_{i=1}^{m} n_i = \sum_{i=0}^{m-1} K^i \qquad \text{(Eq. 4)}$$

$$\lambda \sum_{i=0}^{m-1} iK^i = \sum_{i=0}^{m-1} i\, n_{i+1} \qquad \text{(Eq. 5)}$$

After some algebraic manipulation the following equation is obtained:

$$\frac{(1 - K^m)(1 - K)}{K+(m-1)K^{m+1} - mK^m} = \frac{\sum_{i=1}^{m} n_i}{\sum_{i=0}^{m-1} i n_{i+1}} \qquad \text{(Eq. 6)}$$

Then,

$$\hat{\lambda} = \frac{\sum_{i=1}^{m} n_i}{\sum_{i=0}^{m-1} K^i} \qquad \text{(Eq. 7)}$$

where m = the total number of time intervals (Def. 4)

n_i = the number of errors detected in the i^{th} debugging interval. (Def. 5)

Note: Sukert, in [114], incorrectly defined n_i as the cumulative number of errors detected up through the i^{th} time interval.

Now, referring back to the hazard rate for the Geometric-Poisson model (Eq. 1) it can be shown that this model reduces to Method 1 of Schneidewind's model and thus produces identical results. Below we will prove that the Geometric-Poisson model is in fact equivalent to Schneidewind's model (Method 1).

After comparing numerical results from the two models, we discovered that:

$$K = e^{-\beta} \qquad \text{(Eq. 8)}$$

Thus, the hazard function of the Geometric-Poisson ($Z(t_i) = \lambda K^{i-1}$) may be rewritten as:

$$Z(t_i) = \lambda e^{-\beta(i-1)} \qquad \text{(Eq. 9)}$$

Hence, if we equate $Z(t_i)$ (the hazard rate for the $i^{\underline{th}}$ time interval of the Geometric-Poisson model) with m_i (the estimated number of errors for interval i of Schneidewind's Method 1) we have:

$$\lambda K^{i-1} = (X)[e^{-\beta(i-1)} - e^{-\beta i}] \qquad \text{(Eq. 10)}$$

where $X = (\alpha/\beta)$

Using equation 8, equation 10 may be rewritten as:

$$\lambda e^{-\beta(i-1)} = (X)[e^{-\beta(i-1)} - e^{-\beta i}] \qquad \text{(Eq. 11)}$$

Dividing through by $e^{-\beta(i-1)}$ yields

$$\lambda = X - \frac{X e^{-\beta i}}{e^{-\beta(i-1)}} \qquad \text{(Eq. 12)}$$

$$\lambda = X - X e^{-\beta} \qquad \text{(Eq. 13)}$$

$$\lambda = (\alpha/\beta) - (\alpha/\beta)(e^{-\beta}) \qquad \text{(Eq. 14)}$$

or

$$\lambda = (\alpha/\beta)[1 - e^{-\beta}] \qquad \text{(Eq. 15)}$$

Thus, by substituting equations 8 and 15 in for K^{i-1} and λ respectively we find that the Geometric-Poisson hazard function (equation 1) is equivalent to m_i of the Schneidewind model and equations 16 and 17 support this statement.

$$Z(t_i) = (\alpha/\beta)(1 - e^{-\beta})[e^{-\beta(i-1)}] \qquad \text{(Eq. 16)}$$

or

$$Z(t_i) = (\alpha/\beta)[e^{-\beta(i-1)} - e^{-\beta i}] \qquad \text{(Eq. 17)}$$

which is equivalent to

$$Z(t_i) = \lambda K^{i-1} \qquad \text{(Eq. 18)}$$

Of course we could prove this reduction working in the opposite direction. Since,

$$K^{i-1} = e^{-\beta(i-1)} \qquad \text{(Eq. 19)}$$

$$K = e^{-\beta} \qquad \text{(Eq. 20)}$$

then $\beta = -\ln K$ (Eq. 21)

Then, if we equate $Z(t_i)$ and m_i from the Geometric-Poisson and Schneidewind models respectively we have

$$\lambda K^{i-1} = (\alpha/\beta)\left[e^{-\beta(i-1)} - e^{-\beta i}\right] \quad \text{(Eq. 22)}$$

Substituting equation 21 into equation 22 yields

$$\lambda K^{i-1} = \left(\frac{\alpha}{-\ln K}\right)\left[e^{-(-\ln K)(i-1)} - e^{-(-\ln K)i}\right] \quad \text{(Eq. 23)}$$

$$\lambda K^{i-1}(-\ln K) = \alpha\left[K^{i-1} - K^{i}\right] \quad \text{(Eq. 24)}$$

$$\lambda K^{i-1}(-\ln K) = \alpha K^{i-1}\left[1 - K\right] \quad \text{(Eq. 25)}$$

Now, if we divide both sides of equation 25 by $K^{i-1}[1-K]$ we have an expression for α.

$$\alpha = \frac{\lambda(-\ln K)}{1-K} \quad \text{(Eq. 26)}$$

Thus, if we substitute our expressions for α and β (equations 26 and 21) into the right hand side of equation 22, we obtain:

$$\frac{\frac{\lambda(-\ln K)}{1-K}}{-\ln K}\left[e^{-(-\ln K)(i-1)} - e^{-(\ln K)\,i}\right] \quad \text{(Eq. 27)}$$

Dividing through by -ln K and simplifying yields:

$$\frac{\lambda}{1-K}\left[e^{\ln(K^{i-1})} - e^{\ln(K^{i})}\right] \quad \text{(Eq. 28)}$$

Then,

$$\frac{\lambda}{1-K}\left[K^{i-1} - K^{i}\right] \quad \text{(Eq. 29)}$$

and

$$\frac{\lambda}{1-K}\left[K^{i}\left(\frac{1}{K} - 1\right)\right] \quad \text{(Eq. 30)}$$

which is the same as

$$\frac{\lambda}{1-K}\left[K^{i}\left(\frac{1-K}{K}\right)\right] \quad \text{(Eq. 31)}$$

Dividing through by 1 - K gives

$$\lambda\left[(K^{i})\left(\frac{1}{K}\right)\right] \quad \text{(Eq. 32)}$$

or

$$\lambda\left[(K^{i})(K^{-1})\right] \quad \text{(Eq. 33)}$$

Hence, $$Z(t_i) = \lambda K^{i-1} \quad \text{(Eq. 34)}$$

Thus, $$m_i = Z(t_i) \quad \text{(Eq. 35)}$$

Since the Geometric-Poisson and the Schneidewind model (Method 1) are equivalent, we can work with either model and obtain identical results. Because of the greater flexibility of the Schneidewind model, we will work with it rather than the Geometric-Poisson.

1.6.7 Miyamoto's Revised Shooman Model

In 1973 Isao Miyamoto attempted to expand the basic Shooman model to include MacWilliams' work on input domains [72] and a factor representing errors caused by erroneous debugging. In this model, error detection is a function of the inputs to the system. The input space consists of all possible inputs. During testing, the system is exposed to some subset of the input space; this subset is the test space. Similarly each user will explore a particular portion of what Miyamoto calls the "user space". Errors are triggered or revealed by certain elements of the input space. These elements taken together comprise what might be called an error space. Miyamoto assumes that the error space has a finite initial size for any given program or system, and is expanded only by erroneous debugging. Under this model errors are detected and subsequently corrected wherever the test space overlaps the error space. If the correction is successful, the error space is reduced. Miyamoto defines:

$$Er(\tau) = [E_T + K(\tau)] - Ec(\tau) \quad \text{(Eq. 1)}$$

$Er(\tau)$ = remaining errors at time τ (Def. 1)

E_T = number of initial errors in program (Def. 2)

$K(\tau)$ = number of additional errors due to erroneous debugging (Def. 3)

$Ec(\tau)$ = errors corrected at time τ (Def. 4)

When the test space and error space are disjoint, the system can be said to be fully debugged for that test space. Clearly our concern must be with those inputs

which are in the error space and user space but outside the test space. As exhaustive testing is impossible [13], the test space cannot be expanded to include all of the user space. A great deal of research has been done and reported elsewhere on techniques to select optimal input spaces for software testing and validation. When software is developed it is generally exposed to a test space of steadily growing size. Periodically new inputs are added and new tests are devised. The rate of detection of new errors will be dependent on the rate of increase in the size of the test space. At the same time, if the test space expands in the right "direction", the number of latent errors and the hazard rate for the user space will be reduced. The relationship between the various spaces is diagrammed in Figure 12.

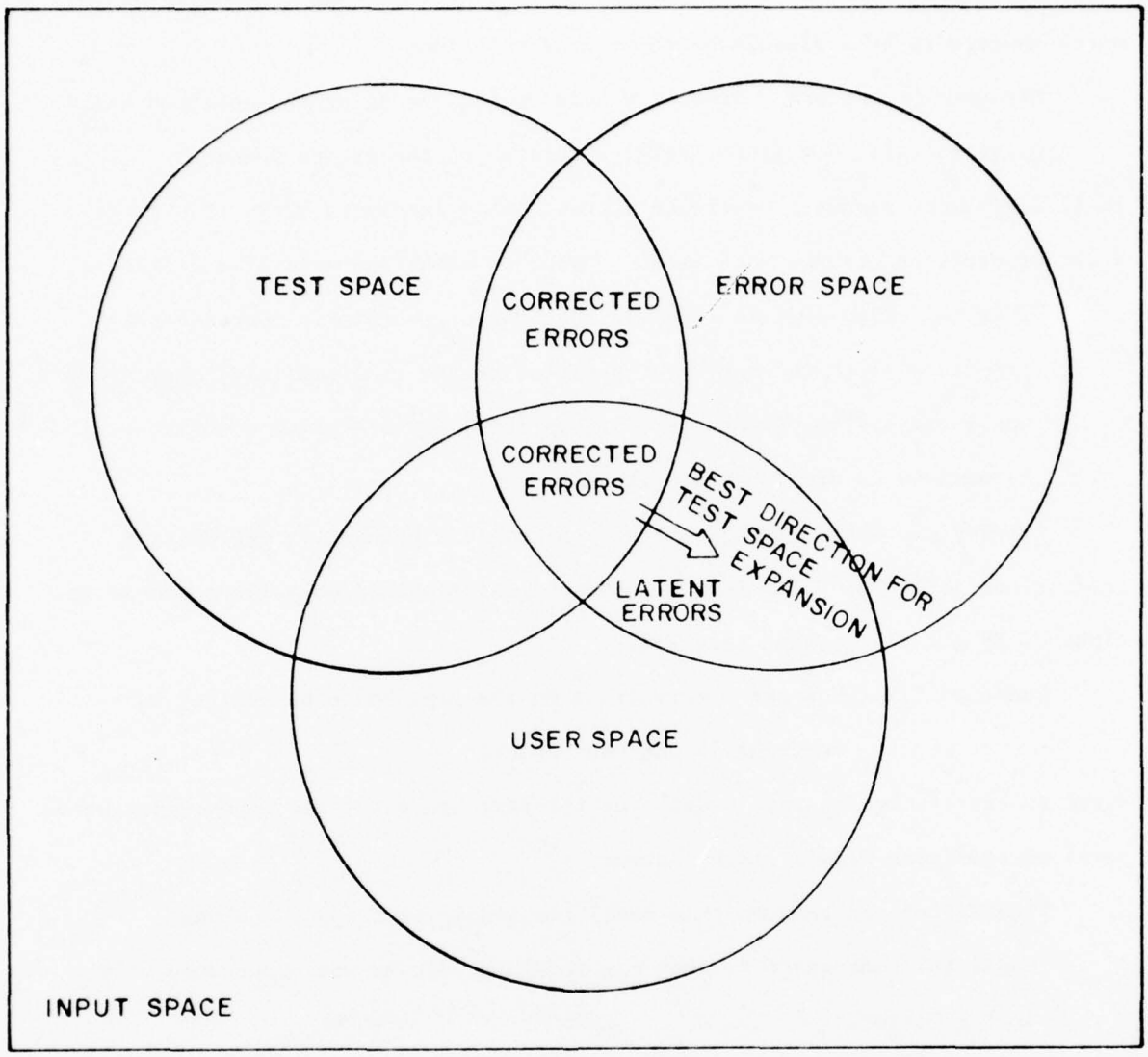
This is the most extensive effort made to date to examine underlying mechanisms in software reliability. As a conceptual model it does an excellent job of describing the causes of the software behavior that the analytical models as a class attempt to predict. Unfortunately, before a numerical model can be constructed several questions must be answered.

(1) If a model is made for the test space, how is reliability related to the errors present? (By definition of the test space you have detected all of the errors which are present, but how do the uncorrected errors affect reliability; how often will you re-encounter them?)

(2) How can observations made in the test space be used to predict behavior in the user space?

(3) How will the reliability estimate be affected if errors in the test space go undetected, particularly errors caused by erroneous debugging?

By combining the solutions to those questions a model would be developed that would relate error detection experience in the growing test space to the population, or incidence, of errors in the user space and then project the effect of



RELATIONSHIP OF TEST SPACE,
USER SPACE AND ERROR SPACE

FIG. 12

those errors on user-encountered reliability. Unfortunately at this point no model appears to be available based on good solutions.

Miyamoto presents a reliability model using the solutions outlined below:

Question (1): How is reliability related to the errors present?

In [77] Miyamoto presents as Figure 9 (reproduced in Figure 13 in this paper) a graph of error occurrence rate versus number of remaining errors, and states:

> "This may allow [us] to conclude that the error rate is approximately proportional to the number of observed errors remained [sic] uncorrected, where the system operates stationary [sic] and the error occurrence is assumed to be distributed uniformly."

Reading the data off Figure 13 we calculate a population correlation coefficient of .4971. This is a significant relationship only if we accept an alpha or Type I risk larger than 10%.

Question (2): How can observations in the test space be related to behavior in the user space?

Miyamoto constructs separate models for the test space and the input space both based on remaining errors, then states:

> "The difference between this model for [the] input space and the model for test space is that the remaining errors are observable only in the latter."

No method is proposed to estimate E_T, initial latent errors. When Miyamoto applies his model to a large online system with six outstanding errors not yet corrected and gets a MTTE of 396.5 hours, he is implicitly assuming that all possible user inputs have been tested (test space = user space), that no other errors exist, and that the linear relationship described in question 1 holds. Clearly, this implies that once those last six errors are fixed the MTTE will be infinite, and the system will never fail.

CORRELATION BETWEEN NUMBER OF REMAINING ERRORS
VERSUS ERROR OCCURRENCE RATE λ_S

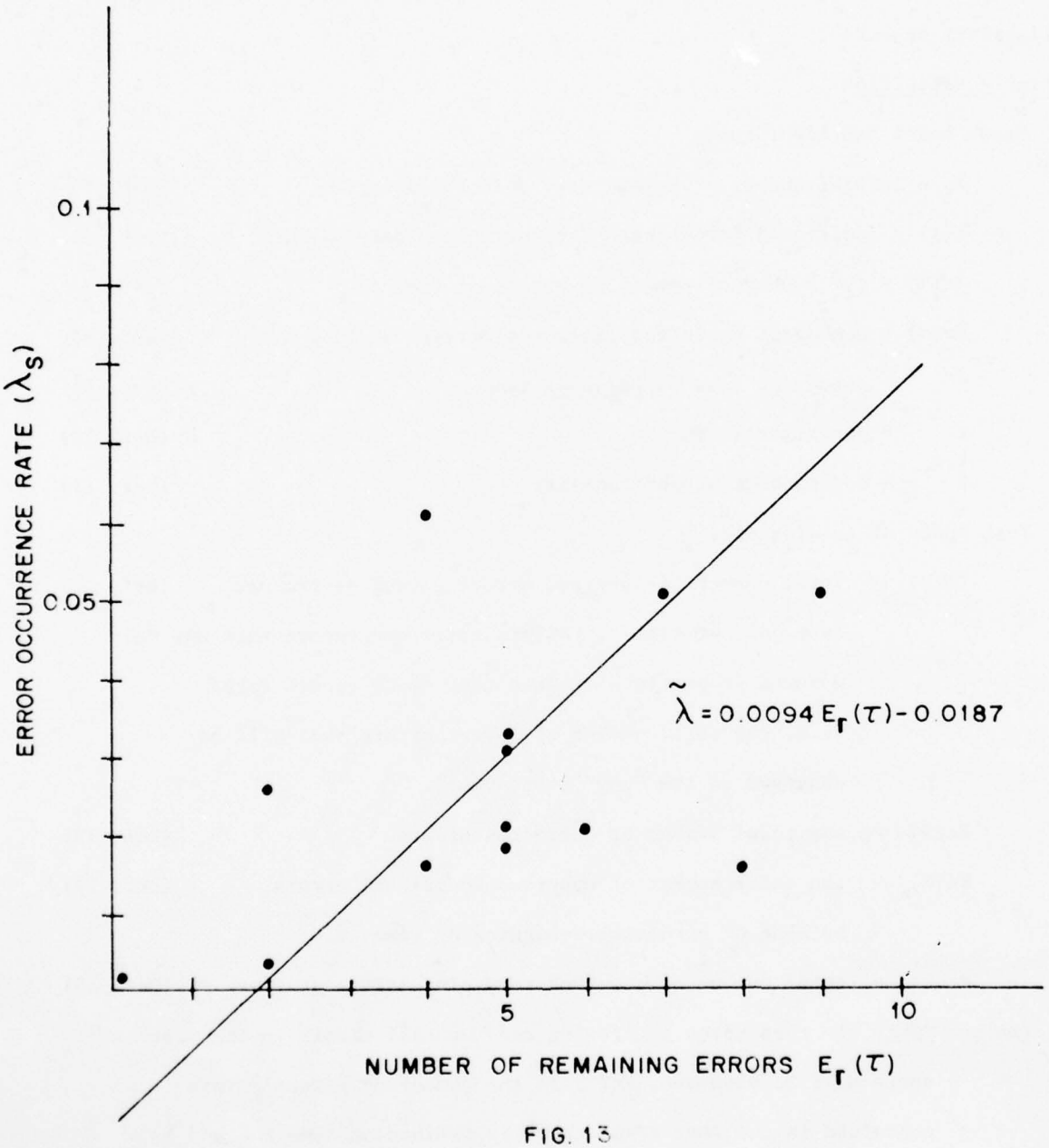


FIG. 13

Thus, although Miyamoto has constructed a good conceptual model of the software "de-eutrophication" process, we feel that his analytical model contains a number of questionable assumptions that render it unsuitable for our purposes. We suspect that this conceptual model may in fact be too detailed to serve as the basis of an analytical model given the current state of the art in software reliability research.

Variable Definition

A. Error model for input space

E_T = initial number of latent errors in the program. (Def. 5)

$K(\tau)$ = additional errors caused by erroneous debugging. (Def. 6)

$Er(\tau)$ = the number of remaining errors at time τ. (Def. 7)

$Ec(\tau)$ = number of corrected (detected) errors at time τ. (Def. 8)

τ = debugging time in calendar time. (Def. 9)

t = operational time. (Def. 10)

C = constant of proportionality. (Def. 11)

B. Test space of growing size

$Eo(\tau)$ = total number of observed errors caused by the new test jobs at time τ (This error occurrence rate may be assumed proportional to the test space growth rate) [i.e. the total number of latent errors that will be observed in the "new" test space]. (Def. 12)

$Ec(\tau)$ = the total number of corrected errors. (Def. 13)

$Ko(\tau)$ = the total number of observed additional errors because of erroneous debugging at time τ. (Def. 14)

$Ero(\tau)$ = total number of observed remaining errors at time τ. (Def. 15)

(Note: Since the test space is growing and since all errors in the test space will be observed, $Eo(\tau)$ is the number of latent errors contained in the test space which is defined at time τ and is not the number of errors actually observed at time τ).

C. Test space of fixed size

E_{To} = initial number of latent errors to be observed in the program (i.e. $Eo(\tau) = E_{To}$ which is a constant). (Def. 16)

$Ko(\tau)$ = total number of observed additional errors caused by erroneous debugging. (Def. 17)

$Ero(\tau)$ = total number of observed remaining errors in the fixed test space. (Def. 18)

Required Test Data

Sequence of times between errors over the entire test space.

Assumptions of Miyamoto's Model

(1) The errors which exist in the "test" space will be actually observed.

(2) The number of errors in a program (at the start of testing) is a constant and decreases directly as errors are corrected.

(3) However, erroneous debugging does introduce new errors.

(4) The failure rate is proportional to the number of residual errors.

(5) It is implicitly assumed that each error has an equal chance of being detected.

(6) The error model developed for the "test space" roughly parallels the appropriate model for "input space" (or at least, the "user space").

(7) The test space can be brought to coincide with the user space in order to exhibit a high degree of software reliability to the user.

-- The "input space" is the set of all possible inputs to the system.

-- The "test space" is the subset of the input space which can be "seen" by the testing personnel (i.e. that subset of the input space actually used during testing).

-- The "user space" is the subset of the input space which is utilized by the user.

Hazard Function

(1) For input space the error model is:

$$Er(\tau) = [E_T + K(\tau)] - Ec(\tau) \quad \text{(Eq. 2)}$$

Then, the hazard function is:

$$\lambda(t) = C[E_T + K(\tau)] - Ec(\tau) \quad \text{(Eq. 3)}$$

(2) For test space of growing size the error model is:

$$Ero(\tau) = [Eo(\tau) + Ko(\tau)] - Ec(\tau) \quad \text{(Eq. 4)}$$

Then, the hazard function is:

$$\lambda(\tau) = C[Eo(\tau) + Ko(\tau)] - Ec(\tau) \quad \text{(Eq. 5)}$$

(3) For test space of fixed size the error model is:

$$Ero(\tau) = [E_{To} + Ko(\tau)] - Ec(\tau) \quad \text{(Eq. 6)}$$

The corresponding hazard function is:

$$\lambda(t) = C[E_{To} + Ko(\tau)] - Ec(\tau) \quad \text{(Eq. 7)}$$

Reliability Equations

(1) For a test space of growing size:

$$R(t,\tau) = \exp\ (\ -\ C[Ero(\tau)]t) \quad \text{(Eq. 8)}$$

$$= \exp\ (\ -\ C[Eo(\tau) + Ko(\tau) - Ec(\tau)]t) \quad \text{(Eq. 9)}$$

(2) For a test space of fixed size:

$$R(t,\tau) = \exp\ (\ -C[Ero(\tau)]t) \quad \text{(Eq. 10)}$$

$$= \exp\ (\ -C[\{E_{To} + Ko(\tau)\} - Ec(\tau)]t) \quad \text{(Eq. 11)}$$

(3) For input space

$$R(t,\tau) = \exp\ (\ -C\{Er(\tau)\}t) \quad \text{(Eq. 12)}$$

$$= \exp\ (\ -C[\{E_T + K(\tau)\} - Ec(\tau)]t) \quad \text{(Eq. 13)}$$

MTTE (MTBSE - Mean Time Between Software Errors)

(1) For a test space of growing size:

$$MTBSE = \frac{1}{C\{Ero(\tau)\}} \quad \text{(Eq. 14)}$$

$$= \frac{1}{C[\{Eo(\tau) + Ko(\tau)\} - Ec(\tau)]} \quad \text{(Eq. 15)}$$

(2) For a test space of fixed size:

$$MTBSE = \frac{1}{C\{Ero(\tau)\}} \qquad \text{(Eq. 16)}$$

$$= \frac{1}{C[\{E_{To} + Ko(\tau)\} - Ec(\tau)]} \qquad \text{(Eq. 17)}$$

(3) For input space:

$$MTBSE = \frac{1}{C\{Er(\tau)\}} \qquad \text{(Eq. 18)}$$

$$= \frac{1}{C[\{E_T + K(\tau)\} - Ec(\tau)]} \qquad \text{(Eq. 19)}$$

1.6.8 Manpower Limited Model

The Manpower Limited model proposed by Shooman and Natarajan in [113] is an attempt to relax the assumption that the number of errors in a program remains constant. The hazard function, equal to the error detection rate, is an input to this model and is assumed to be constant. Instead, this model is formulated to yield an expression for $n(\tau)$, the number of errors remaining in the software. The authors commence by initially developing a difference equation for the number of errors in the program:

Errors present at time τ_i = [errors present at time τ_{i-1}] (Eq. 1)
+ [errors generated in the interval $(\tau_i - \tau_{i-1})$]
- [errors removed in the interval $(\tau_i - \tau_{i-1})$]

If we let:

(1) $n_g(\tau_i, \tau_{i-1})$ = the number of errors generated in the interval $(\tau_i - \tau_{i-1})$ (Def. 1)

(2) $n_d(\tau_i, \tau_{i-1})$ = the number of errors detected in the interval $(\tau_i - \tau_{i-1})$ (Def. 2)

(3) $n_c(\tau_i, \tau_{i-1})$ = the number of errors corrected in the interval $(\tau_i - \tau_{i-1})$ (Def. 3)

then the above difference equation may be written symbolically as:

$$n(\tau_i) = n(\tau_{i-1}) + n_g(\tau_i, \tau_{i-1}) - n_c(\tau_i, \tau_{i-1}) \qquad \text{(Eq. 2)}$$

This difference equation is then transformed into a differential equation by grouping terms, dividing both sides by $(\tau_i - \tau_{i-1}) \equiv \Delta\tau$ and then taking limits, resulting in:

$$\frac{dn(\tau)}{d\tau} = r_g(\tau) - r_c(\tau) \qquad \text{(Eq. 3)}$$

where $r_g(\tau_i)$ = the generation rate of new errors in time τ_i (Def. 4)

$r_c(\tau_i)$ = the correction rate of new errors at time τ_i (Def. 5)

$r_d(\tau_i)$ = the detection rate of errors at time τ_i. (Def. 6)

This differential equation can then be solved for n if the appropriate rates are known.

The basic assumptions associated with the Manpower Limited model are listed below with a short discussion following.

(1) There is a finite number of initial latent errors in a program.

(2) New errors may be generated during debugging.

(3) The error correction rate remains constant during the early stages of debugging, but later decreases, proportional to the number of remaining errors.

(4) The error generation rate is proportional to the product of the number of remaining errors and the rate of detected errors.

(5) The error detection rate is constant.

The major assumption of this model is that the correction rate $[r_c(\tau_i)]$ remains constant during the early stages of debugging (manpower limited) and is decreasing, proportional to the number of remaining errors, in later stages. This assumption was made in an attempt to reflect that which has been practice in the real world and does seem reasonable. During the early stages of testing the manpower for debugging is limited, and the number of errors is large enough to keep them constantly employed, thus producing a constant correction rate. Later on however, fewer errors appear so the debugging manpower may be cut back. Even

if this is not the case, we intuitively feel that later occurring bugs will be harder to fix, resulting in a decreasing correction rate.

Although we felt that the aforementioned assumption is realistic we do have qualms about the assumptions concerning the error detection and generation rates. The error detection rate is assumed to be constant for the entire program, thus severely limiting the applicability of the model. It appears that this assumption was one of convenience since one would expect the detection rate to decrease with time. The second questionable assumption is that the error generation rate is proportional to the product of the number of remaining errors and the rate of error detection. We see no viable reason why it should be proportional to the number of remaining errors and question its relation to the detection rate. At any rate, we also disagree with this assumption as it does not seem to reflect reality. Finally, we felt that this model was not appropriate regarding reliability estimation primarily because the failure rate is required as an input.

Although the Manpower Limited model is not applicable in our research effort a thorough discussion of this model and its requirements may be found on pages 155-170 of reference [113].

1.6.9 Musa Model

John D. Musa of Bell Laboratories has developed a model, based in part on the Shooman and Jelinski-Moranda models, which is described in [85]. Recently he has also made available the user's guide [84] and programming manual [83] for a computer program which implements his model. The Musa model is organized as two sections, one dealing with reliability as a function of execution time and the other relating execution time to calendar time.

The execution time component of the model is based on assumptions very similar to those of the Jelinski-Moranda de-eutrophication model and the Shooman exponential model. The resulting models, like the other two mentioned, have hazard functions that are linear functions of remaining errors.

Musa defines in [85]:

$$T = T_o \exp\left(\frac{c\tau}{M_o T_o}\right) \qquad \text{(Eq. 1)}$$

$$T_o = \frac{1}{fkN_o} \qquad \text{(Eq. 2)}$$

$$M_o = N_o/B \qquad \text{(Eq. 3)}$$

$$M = M_o\,[1-\exp(BCfK\tau)] \qquad \text{(Eq. 4)}$$

where

T = mean time between failures (Def. 1)

T_o = MTBF at start of testing (Def. 2)

C = testing compression factor (Def. 3)

τ = execution time to date (Def. 4)

M_o = number of failures required to expose all N_o (Def. 5)

f = linear execution frequency (Def. 6)

K = error exposure ratio (Def. 7)

B = error reduction factor (Def. 8)

N_o = number of inherent errors (Def. 9)

m = errors found to date (Def. 10)

by substituting equations 2 and 3 into equation 1 we get:

$$T = T_o \exp(BCfk\tau) \qquad \text{(Eq. 5)}$$

Equation 4 can be rewritten as:

$$\exp(BCfk\tau) = M_o/(M_o - m) \qquad \text{(Eq. 6)}$$

Substituting equation 6 into equation 5 gives:

$$T = \frac{T_o M_o}{M_o - m} \qquad \text{(Eq. 7)}$$

If we let

$$\phi = Bfk = \frac{1}{T_o M_o} \quad \text{(a constant)} \qquad \text{(Eq. 8)}$$

$$N = M_o \quad \text{(initial errors)} \qquad \text{(Eq. 9)}$$

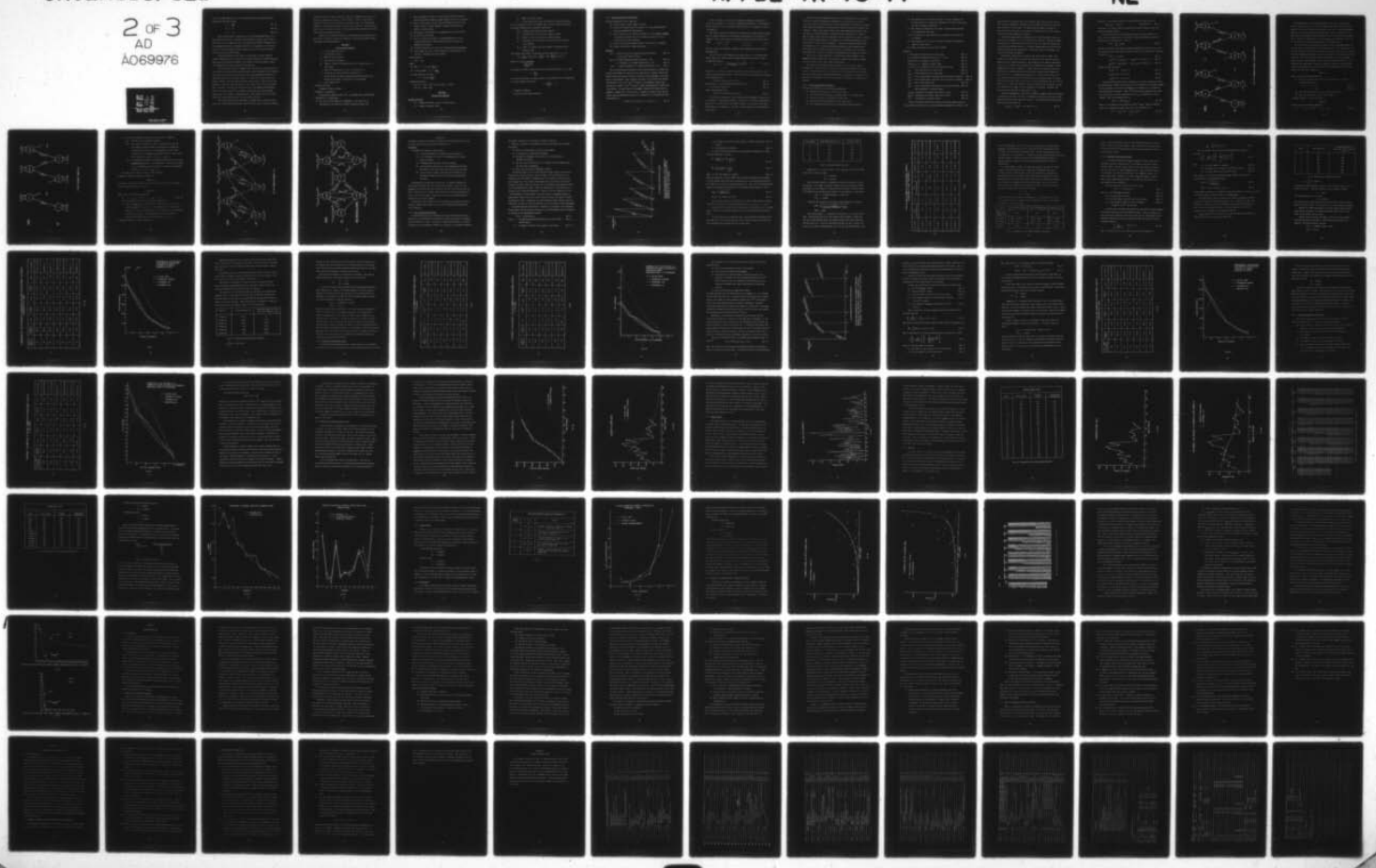
$$i-1 = m \quad \text{(errors to date)} \qquad \text{(Eq. 10)}$$

Equation 7 becomes:

$$T = \frac{1}{\phi\,[N-(i-1)]} \qquad \text{(Eq. 11)}$$

AD-A069 976 DAYTON UNIV OHIO F/G 9/2
SOFTWARE RELIABILITY: DETERMINATION AND PREDICTION.(U)
JUN 78 L S GEPHART, C M GREENWALD F33615-77-C-3072
UNCLASSIFIED AFFDL-TR-78-77 NL

2 OF 3
AD
A069976



This is the MTBF expression for the Jelinski-Moranda de-eutrophication model. Working the other way we find:

$$T_o = \frac{1}{\emptyset N} \qquad \text{(Eq. 12)}$$

$$M_o = N \qquad \text{(Eq. 13)}$$

$$m = i-1 \qquad \text{(Eq. 14)}$$

While the Musa model is equivalent to the Jelinski-Moranda model, it is much harder to work with. The model used 9 different constants, most of which Musa says cannot be computed and must be estimated from "similar programs", then re-estimated periodically by maximum likelihood. For example, it may be noted that in the above derivation B,C,f,K,T_o,M_o,N_o, and m are all various constant factors, rates, and initial values.

Musa goes beyond this basic model to incorporate an "error reduction factor", making the error correction rate a linear function of the detection rate. Using this he then derives estimates of the number of failures needed to detect all remaining errors and the execution time required.

We feel however, that this assumption that effectively a fixed fraction of the errors are corrected before testing resumes, and the remainder must be detected again before they can be corrected is most inaccurate, making the derivation of remaining failures and time highly suspect. It would be far more realistic to assume a definite time lag between detection and correction. Also, we feel that making the correction rate proportional to the failure rate is far less realistic than it could be. We prefer the correction rate of the Manpower Limited model; which is initially constant due to manpower limitations and later decreasing, proportional to the number of remaining errors. This implies that the last bugs found are the most subtle, harder to locate and fix correctly.

The second component of the Musa model, calendar time, is where its great value lies. Musa feels that execution time is the best scale to use with software,

and develops the basic model using that. Musa then recognized that the pace of testing is constrained by limits on three resources: failure identification personnel, failure correction personnel, and computer time. The calendar time component of the Musa model defines limits for these resources, relates execution time to the use of these resources, then estimates how much calendar time it will take to supply the needed resources.

Because of the similarity between the Jelinski-Moranda and Musa models, we have chosen to pursue our current investigation of error detection models using the simpler Jelinski-Moranda model.

Musa Model

Execution Time Component

C = testing compression factor

B = error reduction factor

T_o = MTBF of start of testing

τ = execution time of program

N_o = number of inherent (existing before the test phase) errors in the program

η = number of remaining errors = $N_o - n$ (a function of τ)

n = net number of errors corrected (a function of τ)

f = linear execution frequency (average instruction execution rate divided by the number of instructions in the program)

K = error exposure ratio

-- Required test data

Sequence of times to failure

-- Estimation of parameters

Previous data, similar projects (K), η_o "computed from collected data"

-- Assumptions of the Musa model

(1) Any errors in the program are independent of each other and are distributed at any time with a constant average occurrence rate.

(2) Thus, the number of errors in a given time interval has a Poisson distribution (whose parameter changes whenever an error is corrected).

(3) Types of instructions are reasonably well mixed, and execution time between failures is large compared to average instruction time (implicit in most models).

(4) The potential "test space" for the program covers its "use space".

(5) All failures are observed.

(6) The error causing each failure is fixed immediately or it is not recounted (Musa tends to assume implicitly that all errors are immediately corrected).

(7) Initially assumes that no new errors are generated during debugging.

(8) Later assumes that only a fixed percentage of detected errors are corrected. Assumption (6) still applies.

-- Hazard Function

$$Z(\tau) = Kf\eta$$

-- MTBF

(1) $\text{MTBF} = T = T_o \exp\left(\frac{C}{M_o T_o}\tau\right)$

(2) If initial $\text{MTBF} = T_o = \frac{1}{fKN_o}$

(3) Then, equation (1) becomes

$$T = T_o \exp\left(\frac{\tau}{N_o T_o}\right)$$

-- Reliability Equation

$$R(\tau,\tau') = \exp\left[-\int_o^{\tau'} Z(\tau)\, dx\right] = \exp\left[-\tau' Z(\tau)\right]$$

$$R(\tau,\tau') = \exp\left(-\frac{\tau'}{T}\right)$$

Musa Model

Calendar Time Component

Variable definition

M_o = number of failures required to expose all N_o

N_o = number of inherent errors

T_o = MTBF at start of testing

C = testing compression factor (non-overlap of successive tests)

For the folloiwng K = C,F,I for computer time, failure correction personnel, and failure identification personnel, respectively.

Δt_K = calendar time required for this resource

P_K = available units of the resource (people, shifts)

μ_K = amount of time this resource is used per failure

ρ_K = utilization of this resource (e.g. fraction of day actually spent working)

θ_K = amount of calendar time this resource is used per unit of execution time

The calendar time needed to accumulate execution time from τ_1 to τ_2 is:

$$\Delta t_K = \frac{1}{P_K \rho_K} \{ M_o \mu_K [\exp (- \frac{C\tau_1}{M_o T_o}) - \exp (\frac{C\tau_2}{M_o T_o})] + \theta_K \Delta\tau \}$$

where K is chosen to maximize

$$\frac{\theta_K T + C\mu_K}{P_K \rho_K T}$$

If testing is entirely limited by failure-correction personnel, $\theta_F = 0$, and

$$\tau << \frac{M_o T_o}{C}$$

we can easily determine the maximum amount of testing which can be accomplished in a given calendar time period.

$$\Delta\tau = \frac{P_F T_o \rho_F}{C\mu_F} \Delta t$$

-- Parameter estimation

Previous data and similar projects.

1.6.10 Bayesian Reliability Growth Model

The basic assumptions of this model are:

(1) Does not assume a finite number of errors.

(2) Failure rate between errors is constant (i.e. time between failures follows an exponential distribution).

(3) Error correction is not always successful (i.e. the programmer intends to decrease the failure rate in an attempted correction, but the correction may not be successful).

(4) Failure rate declines probabilistically as correction is attempted.

(5) Failure rates follow a Gamma distribution.

Notation:

$\lambda(i)$ = failure rate in the interval between the detection of the $(i-1)^{st}$ and i^{th} errors (Def. 1)

$g(\ell \mid i,\alpha)$ = Gamma probability density function of $\lambda(i)$ (Def. 2)

α = a parameter of the Gamma distribution of $\lambda(i)$ (Def. 3)

$\psi(i)$ = a growth parameter of the Gamma distributions of $\lambda(i)$ (Def. 4)

The so-called Bayesian Reliability Growth model proposed by Littlewood and Verrall [67] attempts to account for error generation by creating a repair rule which as nearly as possible reproduces the effect of the programmer's corrective action on the program. Given a failure rate $\lambda(i-1)$ as the failure rate between the detection of the $(i-1)^{st}$ and i^{th} failures, the programmer intends, when carrying out a repair to the i^{th} failure, to make the program more reliable than it was before. In other words, the programmer intends by his repair action to diminish λ, making $\lambda(i) < \lambda(i-1)$ for all i. However, he cannot be sure that the failure rate has diminished; instead, it is argued that it is probable that this has happened, i.e.

$$P(\lambda(i) < \ell) \geq P(\lambda(i-1) < \ell) \quad \text{for all } \ell, i. \qquad \text{(Eq. 1)}$$

This, in essence, is a resort to the Bayesian technique of allowing the failure rate parameter, $\lambda(i)$, to have a probability distribution. The probability density function of $\lambda(i)$ is denoted by $g(\ell|i,\alpha)$ where α is a parameter or vector of parameters.

The authors propose that failures be considered as a random (Poisson) process, that is, between failures (or within small time intervals) the failure rate is constant. This in turn leads to an exponential density function for the failure times,

$$f(t|\lambda) = \lambda e^{-\lambda t} \qquad t > 0,\ \lambda > 0 \qquad \text{(Eq. 2)}$$
$$= 0 \qquad t < 0,\ \lambda > 0$$

This is a popular and reasonable assumption and is perhaps the most tractable mathematically.

For the failure rates themselves, the authors select, as a "suitable parametric family with a monotonically arranged distribution function", a family of Gamma distributions,

$$g(\ell|\ i,\alpha\ ) = \frac{\psi(i)[\psi(i)\ell\ ]^{\alpha-1}\ e^{-\psi(i)\ell}}{\Gamma(\alpha)} \qquad \ell > 0 \qquad \text{(Eq. 3)}$$
$$= 0 \qquad \ell > 0$$

where $\psi(i)$ is a scaling or "growth of reliability" factor which must be a monotonically increasing function of i. The fact that $\psi(i)$ is monotonically increasing with i guarantees that

$$G(\ell|i,\alpha) \geq G(\ell|(i-1),\alpha) \text{ for all } i,\ \ell \qquad \text{(Eq. 4)}$$

where $G(\ell|i,\alpha)$ is the distribution function of $\lambda(i)$ (Def. 5)

i.e. $G(\ell|i,\alpha) = P(\lambda(i)<\ell)$. (Eq. 5)

Hence, this guarantees that

$$P(\lambda(i)<\ell) \geq P(\lambda(i-1)<\ell) \text{ for all } i,\ell \qquad \text{(Eq. 6)}$$

In other words, the nature of $\psi(i)$ represents the programmer's intention, but not certainty, of improving the program. This choice of a family of Gamma distributions for the failure rates, the authors contend, is justifiable by its flexibility (having two parameters, α and $\psi(i)$), correct range $(0,\alpha)$, and mathematical tractability.

Giving λ this prior (Gamma) distribution is a standard practice in a Bayesian analysis. The prior distribution is supposed to be a reflection of the prior (i.e. before data is collected) beliefs of the experimenter. After data is collected, these beliefs are modified by the data, and the result is given in the posterior distribution. However, the authors deviate from a usual Bayesian analysis when they also give α a prior distribution. Putting the prior distribution on α seems to make the model a great deal more complicated and very difficult to analyze. Furthermore, the prior distribution on α is assumed to be a uniform distribution, and no justification is given for this assumption. Arbitrarily assigning uniform priors to parameters is one of the more criticized techniques in early Bayesian works. The point is that it would be extremely difficult to determine one's true prior distribution for α since it is another step removed from the variable of interest λ .

There is also a major difficulty in estimating the other parameter of the Gamma distribution, namely $\psi(i)$. The authors do consider some estimation methods for $\psi(i)$ but admit that it does present some problems.

The authors do not attempt to apply the model to any real data, which is disappointing. It should also be noted that this model was proposed in 1973, and to date, no follow-up work on it has appeared in the literature. While the model does have intuitive appeal and appears applicable to many realistic situations, the difficulty in estimating α and $\psi(i)$ is too great for the model to be useful in our analyses.

1.6.11 Trivedi and Shooman Markov Model

The basic model (Model I) has the following assumptions:

(1) Software system is large ($\sim 10^5$ words of code).

(2) The system initially contains an unknown number, n, of unknown bugs.

(3) At most one error is discovered at a given time.

(4) Each error is corrected before the next error occurs.

(5) Error detection and correction occur alternately and sequentially.

(6) The probability of transition from state i to state j depends only on those states and is completely independent of all past states except the last one (Markov assumption).

(7) The failure rate depends upon the number of software bugs remaining in or removed from the system.

(8) In the interval of time between error occurrences the failure rate is constant.

(9) The attempted error correction is always successful in reducing the number of errors by one.

(10) There are two types of states "up" or "down".

Notation:

n = the number of initial errors in the system (Def. 1)

P_{ij} = transition probability from state i to state j (Def. 2)

$(n, n-1, n-2, \ldots)$ = sequence of "up" states (Def. 3)

$(m, m-1, m-2, \ldots)$ = sequence of "down" states (Def. 4)

$(\ell, \ell-1, \ell-2, \ldots)$ = sequence of "non critically down" states (Def. 5)

λ_{n-k} = error detection rate in state (n-k) (Def. 6)

μ_{m-k} = error correction rate in state (m-k) (Def. 7)

α_{m-k} = rate of unsuccessful correction and no new errors introduced (Def. 8)

β_{m-k} = rate of unsuccessful correction with one new error introduced (Def. 9)

η_{m-k} = non critical failure rate (Def. 10)

$\phi_{\ell-k}$ = rate of unsuccessful correction of a non critical failure which introduces a critical failure (Def. 11)

$P_{n-k}(t)$ = probability of being in state (n-k) at time t (Def. 12)

$P_{m-k}(t)$ = probability of being in state (m-k) at time t (Def. 13)

$A(t)$ = availability of system at time t (Def. 14)

Trivedi and Shooman [130-132] have proposed a many-state Markov model for the estimation and prediction of various performance parameters of software. This

model depicts the process by which error detection and correction occurs, and is used to predict reliability, availability and the number of errors that will have been corrected at a future time. The model makes no attempt to estimate or describe the error detection rate or the error correction rate. On the contrary, these are required as inputs to the model.

Trivedi and Shooman begin by assuming that the software system is large and contains a fixed number, n, of errors initially (at time t=0). The most meaningful time origin for the model would be the start of Phase II, where the program runs, at least briefly, between errors. It is further assumed that errors occur and are corrected alternately and sequentially. In the basic version of the model, all errors are corrected successfully.

The software system can be in either of two states: "up" or "down". The system is in an "up" state if no error has occurred or if an error has just been repaired (i.e.the system is operable). The sequence of "up" states is denoted by (n, n-1, n-2,...) where the state number can be thought of as the number of errors remaining in the system. Similarly, a "down" state is one in which an error has occurred and is being corrected, and the sequence of down states is denoted by (m, m-1, m-2,...). In general, the system will be in state (n-k) if the $(k-1)^{st}$ error has been corrected and the k^{th} error has not yet occurred, while the system will be in state (m-k) after the k^{th} error has been discovered but not yet corrected (k=0,1,2,..,n). The error occurrence rate while in state (n-k) is denoted λ_{n-k} and is a function of the number of errors remaining in the system. Similarly, the error correction rate while in state (m-k) is μ_{m-k}.

Trivedi and Shooman next invoke the basic Markov assumption that the probability of transition from state i to state j is dependent only on those states and is independent of all past states except the last one. The probability of transition from state (n-k) to state (m-k) is

$$P_{n-k,m-k} = \lambda_{n-k} \Delta t \quad \text{for} \quad k=0,1,2,..,n. \qquad \text{(Eq. 1)}$$

Similarly, the transition probability from state (m-k) to state (n-k+1) is

$$P_{m-k,n-k+1} = \mu_{m-k} \Delta t \qquad \text{(See Figure 14)} \qquad \text{(Eq. 2)}$$

The entire set of these transition probabilities defines a discrete-state, continuous time Markov process.

If we now let $P_i(t)$ be the probability that the system is in state i at time t, then the availability of the system is simply the probability that the system is in any "up" state at t. That is

$$A(t) = \sum_{k=0}^{n} P_{n-k}(t) \qquad \text{(Eq. 3)}$$

In order to determine these "state occupancy probabilities", $P_i(t)$, we must solve the following system of differential equations (which can be developed by inspection of Figure 14):

$$P_n(t) = \lambda_n P_n(t) \qquad \text{(Eq. 4)}$$

$$P_{n-k}(t) + \lambda_{n-k} P_{n-k}(t) = \mu_{m-k+1}(t) \; ; \quad k=1,2,..,n \qquad \text{(Eq. 5)}$$

$$P_{m-k}(t) + \mu_{m-k} P_{m-k}(t) = \lambda_{n-k} P_{n-k}(t) \; ; \; k=0,1,2,..,n \qquad \text{(Eq. 6)}$$

subject to these conditions:

$$P_n(0) = 1; \quad P_m(0) = 0 \qquad \text{(Eq. 7)}$$

$$P_{n-k}(0) = 0; \quad k = 1,2,..,n \qquad \text{(Eq. 8)}$$

$$P_{m-k}(0) = 0; \quad k = 1,2,..,n \qquad \text{(Eq. 9)}$$

Trivedi and Shooman next present techniques for the exact analytical solution and the numerical solution of this system. The reliability of the software system, of course, depends on the stage of debugging of the system (i.e., the number of bugs remaining in the system). After the k^{th} bug has been removed and the system is in state (n-k), the error occurrence rate is λ_{n-k}, a function of k, and is constant. Thus, the reliability will be

$$R_k(\tau) = \exp(-\lambda_{n-k} \tau) \qquad \text{(Eq. 10)}$$

where τ is the time since the correction of the k^{th} error. Hence the expected value of the reliability at some future time t and a duration τ is:

$$R(t,\tau) = \sum_{k=0}^{n} \exp(-\lambda_{n-k}) P_{n-k}(t) \qquad \text{(Eq. 11)}$$

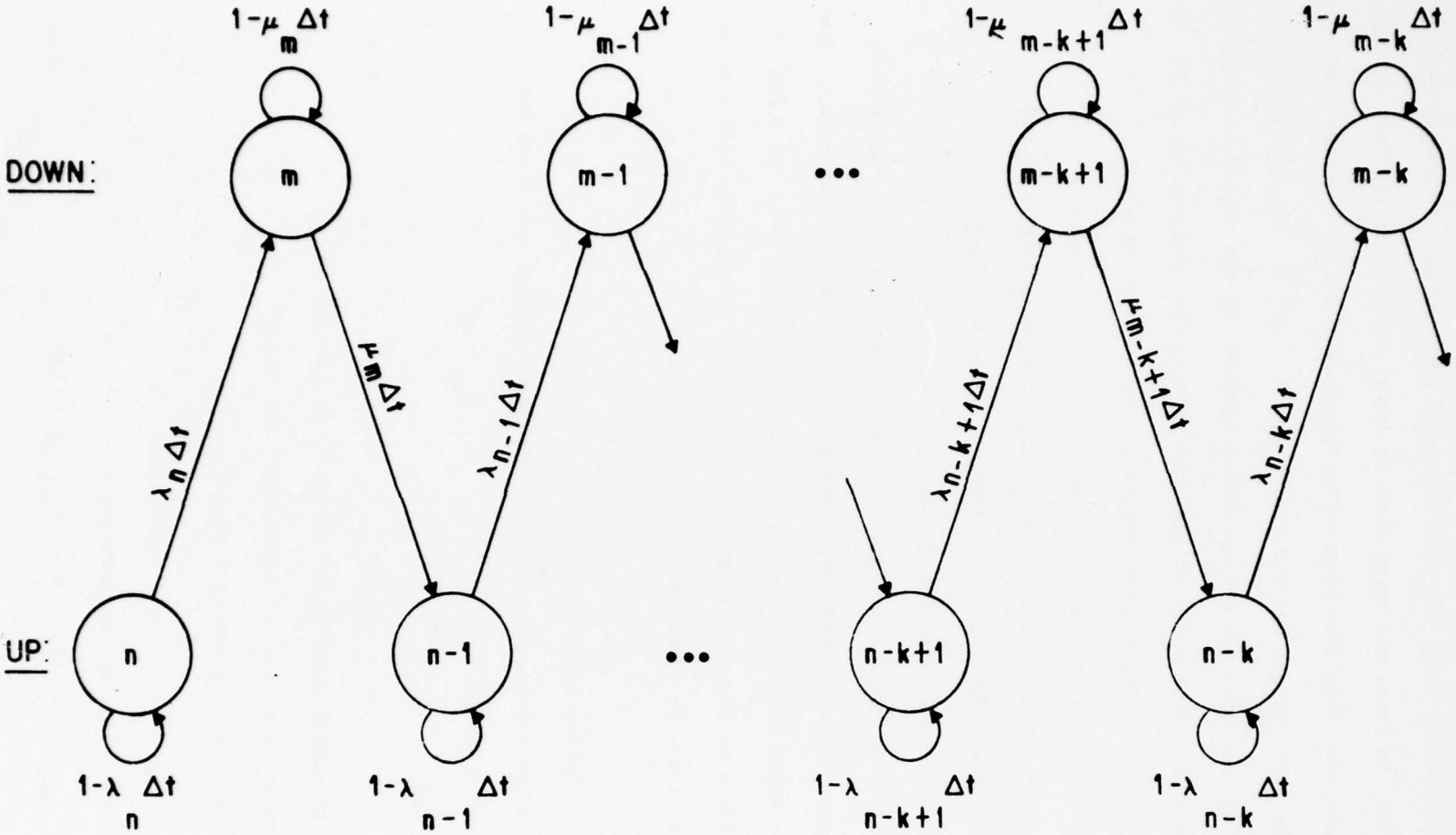


FIG. 14 - MARKOV MODEL I (BASIC MODEL)

This Markov model, then, provides estimates of the state occupancy probabilities which, in turn, can be used to estimate and predict the most probable number of errors that will have been corrected at any time t, based on preliminary modeling of the error occurrence and repair rates, λ and μ. Equivalently, this means that we can estimate the most probable state(s) of the system at some future time t. Given this information, we can then predict the availability, A(t), and the reliability $R(t,\tau)$, for a given duration τ at a future time t.

Up to this point we have discussed in detail only the basic model (Model I). The basic model is characterized by the assumptions that λ and μ depend only on k, that the error correction is always successful, and that there are only two states. Trivedi and Shooman also present a number of variations of the basic model in [131] and [132]. All these variations have the same assumptions as the basic model except the three just mentioned, which are altered in some form or other. These variations are named Model II, Model I-G, Model II-G, Model I-H, and Model II-H. They all provide some increase in generality and thus flexibility over the basic model. They will now be summarized.

Model II

Model II has the following assumptions:

(1) The error occurrence rate λ and the error repair rate μ are both explicit functions of t.

i.e. $\lambda = \lambda(t)$ (Def. 15)

$\mu = \mu(t)$ (Def. 16)

(2) The attempted correction of code is always successful.

(3) There are two types of states "up" and "down".

A diagram of Model II is given in Figure 15.

Model I-G

Model I-G has the following assumptions:

(1) $\lambda = \lambda_{n-k} = \lambda(k)$; $\mu = \mu_{m-k} = \mu(k)$ (Def. 17)

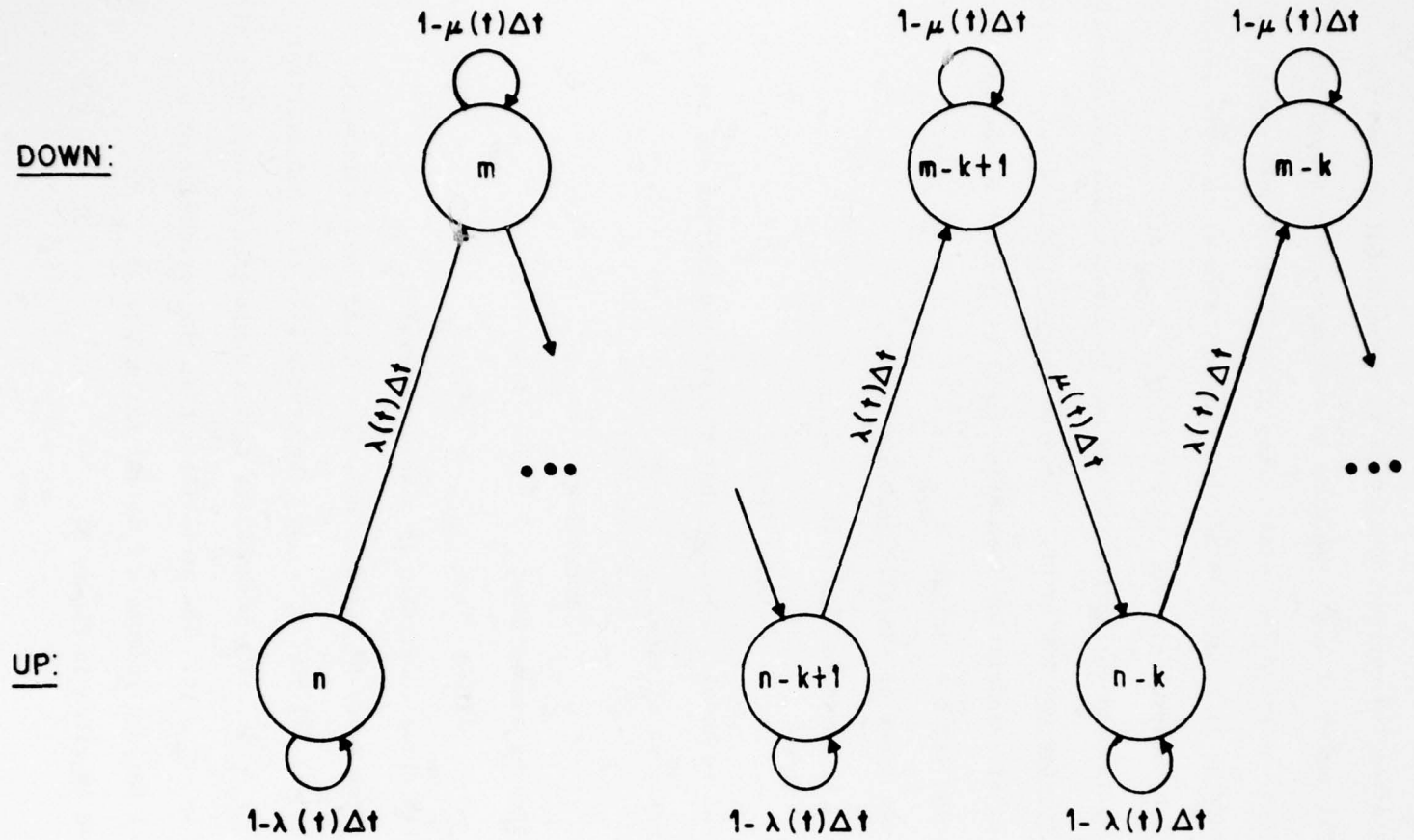


FIG. 15 - MARKOV MODEL II

(2) One of three things can happen when a correction is attempted:

(i) the bug can be successfully repaired,

(ii) the attempted correction of code is unsuccessful so that the total number of bugs remaining is unchanged, even though the system enters an "up" state. The probability of this event occurring (i.e. of transmitting from state (m-k) to state (n-k)) in the interval of time $(t, t + \Delta t)$ is $\alpha_{m-k} \Delta t$,

(iii) the attempted correction of code may be unsuccessful and actually introduces one new error. The probability of this event occurring (i.e. of transmitting from state (m-k) to (n-k+1) in the interval of time $(t, t + \Delta t)$ is $\beta_{m-k} \Delta t$.

(3) There are two types of states: "up" and "down".

A diagram of Model I-G is given in Figure 16.

Model II-G

This model is identical to Model I-G except that the error detection and repair rates are explicit functions of time.

Model I-H

Model I-H has the following assumptions:

(1) $\lambda = \lambda_{n-k} = \lambda(k)$; $\mu = \mu_{m-k} = \mu(k)$. (Def. 18)

(2) The attempted error correction is always successful.

(3) There are three types of states: "up", "down", and "non critically down" (degraded) $(\ell, \ell-1, \ell-2, \ldots)$. The probability of a non critical failure is $\gamma_{n-k} \Delta t$. The probability that a correction is only partially successful is $\eta_{m-k} \Delta t$. The probability that the correction of a non-critical failure induces a critical failure is $\phi_{\ell-k} \Delta t$.

A diagram of Model I-H is given in Figure 17.

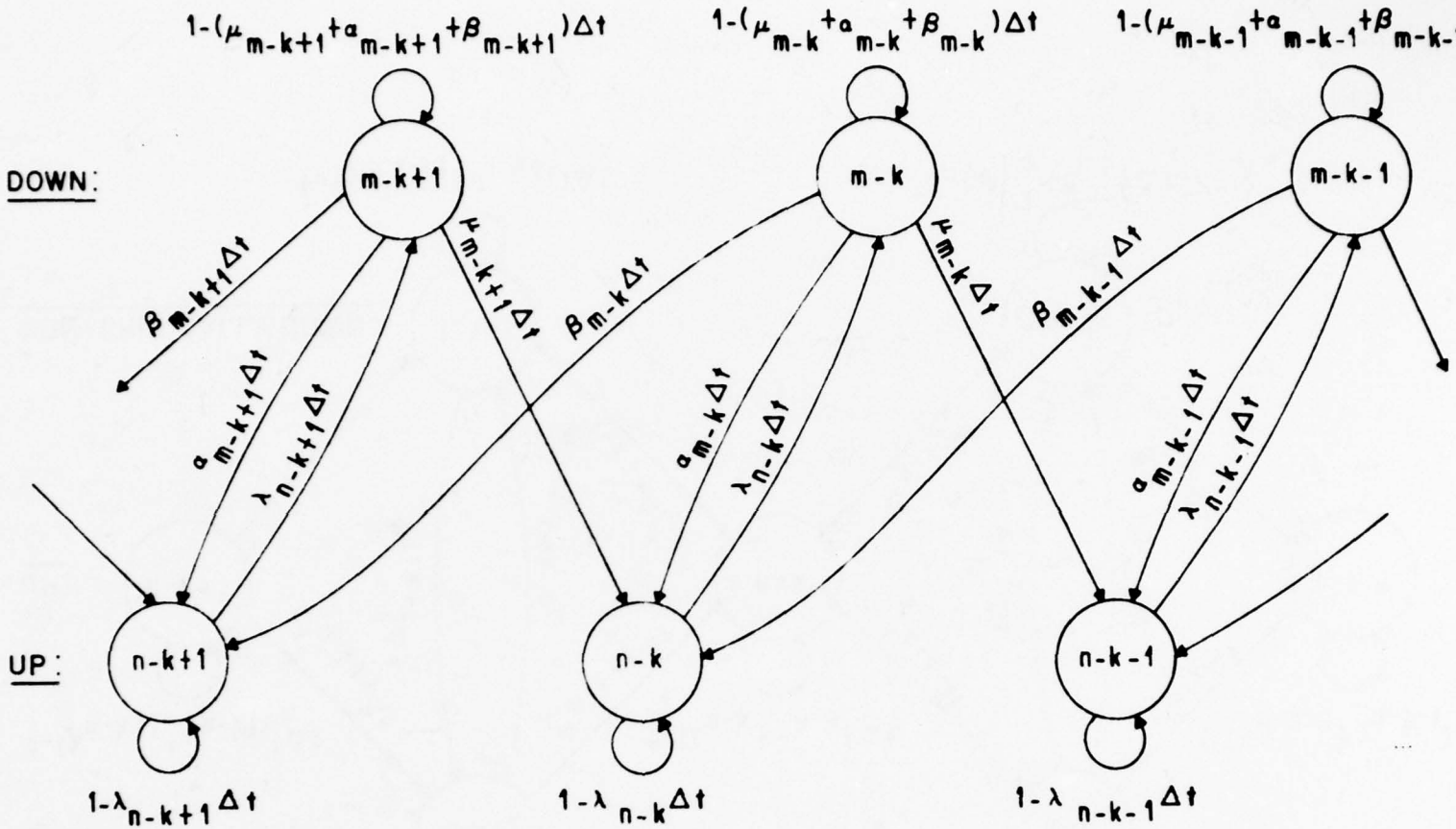


FIG. 16- MARKOV MODEL I-G

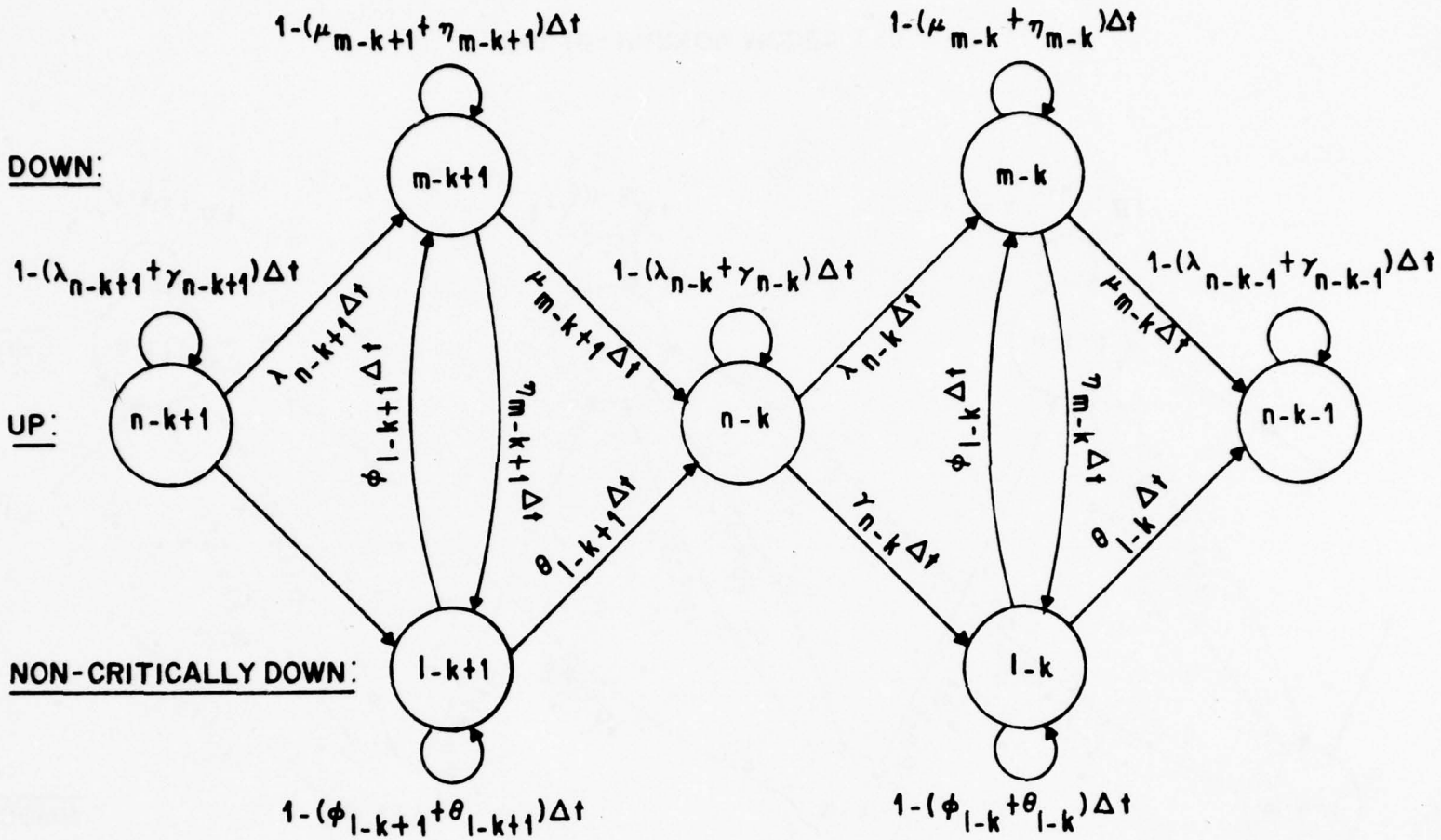


FIG.17- MARKOV MODEL I-H

Model II-H

This model is identical to Model I-H, except that the error detection rate and the error correction rate are explicit functions of t.

Conclusions

This model has a number of strong points:

(1) It provides estimates of reliability and availability.

(2) It is compatible with many of the assumptions of the other analytical models.

(3) It mathematically describes the error behavior.

(4) It realistically describes the entire debugging process when the detection rate (λ) and the correction rate (μ) are known.

(5) It may be able to handle a multiple state condition for an operational software system if the transition probabilities are known.

Unfortunately, the model has one key flaw. It presents no method for determining the failure rate (λ) of the software package. In fact, it even requires the error detection rate (as well as the error correction rate) as an input. Our effort is to actually find a method for accurately determining the failure rate of the software. Therefore, we do not feel the Markov model is applicable to our current analysis.

Downstream it may be possible to use this model in conjunction with other analytical models which provide λ and μ so that reliability and availability can be determined.

1.6.12 Basic Schick-Wolverton Model

The Schick-Wolverton model, developed by George Schick and Ray Wolverton, modifies the Jelinski-Moranda de-eutrophication model by changing the assumption about the behavior of errors. Initially, a listing and discussion of the model assumptions will be presented, followed by the equations for parameter estimation

and finally, application of this model to software error data.

Below is a listing of the assumptions Schick and Wolverton make regarding their model.

(1) There is a fixed number of errors in the program.

(2) No new errors are added during debugging.

(3) The amount of debugging time between error occurrences has a Rayleigh distribution.

(4) The error rate is proportional to the number of errors remaining and the time spent in debugging.

(5) Each error discovered is immediately removed.

As mentioned above, the basic Schick-Wolverton model assumes that the error detection rate is proportional to both the number of errors remaining in the software and the time spent in debugging since the detection of the most recent error (Figure 18). This implies that between errors, the failure rate actually increases with time, using the rationale that the program's inputs gradually close *in on the remaining errors [86]. Under this assumption,* the time between error occurrences has a Rayleigh distribution. However, we do not consider this to be a valid assumption, except perhaps in the early stages of debugging (Figure 1). Beyond Phase I however, we do not believe that the failure rate would be linearly increasing with time. Furthermore, the Schick-Wolverton model indicates that the hazard rate is zero just after the detection of an error, which is also questionable.

The hazard function, given below, is identical to that given in the Jelinski-Moranda model, with the exception being that t_i (the time between error occurrences) is included in the Schick-Wolverton model.

$$Z(t_i) = \emptyset\,[N-(i-1)]\,t_i \qquad \text{(Eq. 1)}$$

where t_i = the $i^{\underline{th}}$ time interval between detection of the $(i-1)^{\underline{st}}$ and $i^{\underline{th}}$ errors. (Def. 1)

N = the number of initial errors present in the system. (Def. 2)

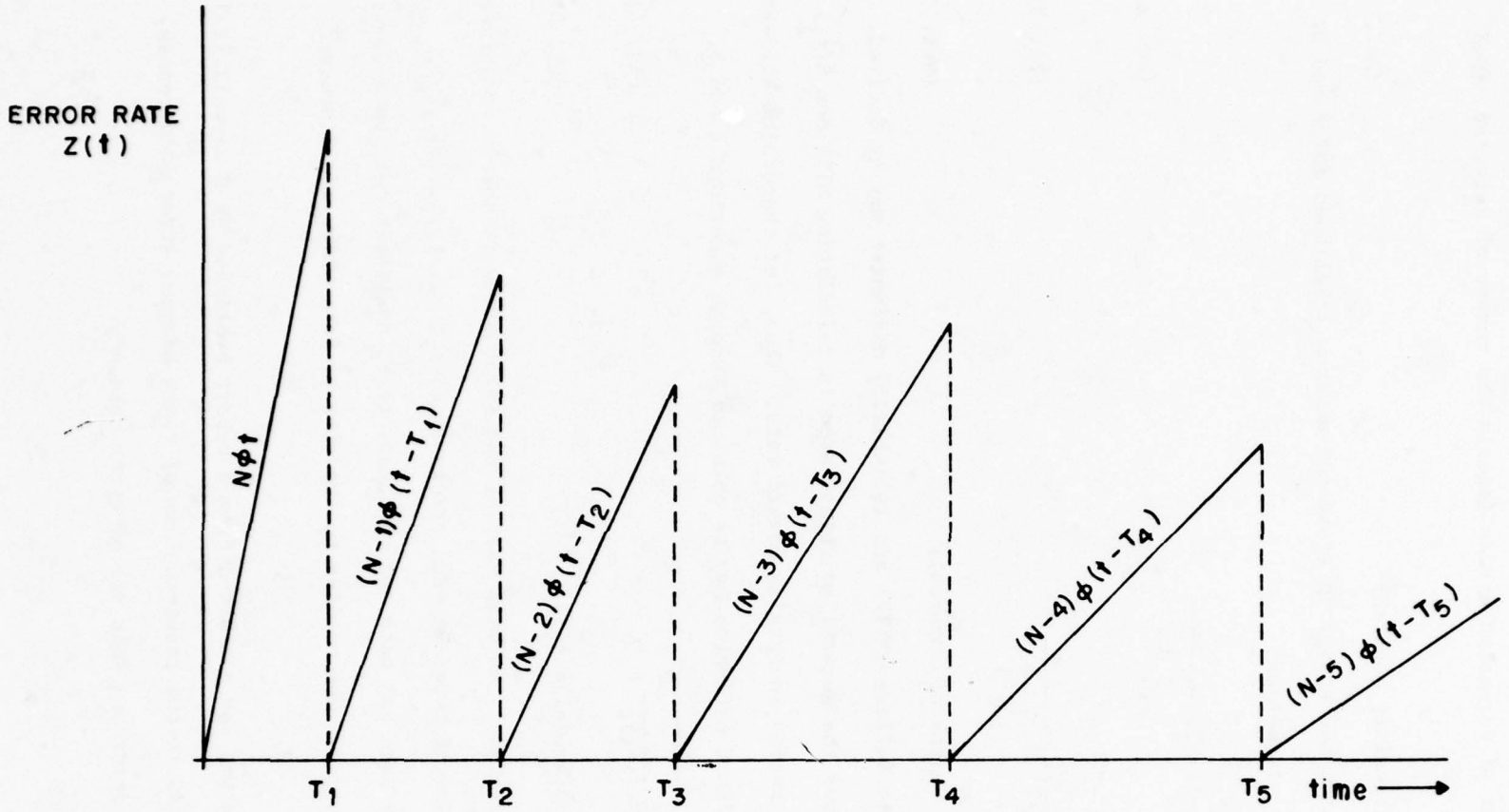


FIG. 18-SCHICK-WOLVERTON MODEL

FAILURE RATE PROPORTIONAL TO THE NUMBER OF REMAINING ERRORS AND THE TIME SINCE THE DETECTION OF THE LAST ERROR.

i = the number of corrected errors (equals the number of detected errors). (Def. 3)

$\emptyset$ = a proportionality constant (Def. 4)

In [138], Schick and Wolverton state the maximum likelihood for N and $\emptyset$, the model parameters, as being:

$$\hat{N} = \left[\frac{2n}{\emptyset} \sum_{i=1}^{n} (i-1)\ t_i^2\right] \frac{1}{\sum_{i=1}^{n} t_i^2} \qquad \text{(Eq. 2)}$$

$$\hat{\emptyset} = \left[\sum_{i=1}^{n} \frac{2}{N-(i-1)}\right] \frac{1}{\sum_{i=1}^{n} t_i^2} \qquad \text{(Eq. 3)}$$

where n = the total number of intervals (Def. 5)

Then, the mean time to failure (MTTF) and reliability estimates may be derived. In [111], Shooman gives the general equations used in calculating MTTF and $R(t_i)$ for a model with a linearly increasing hazard rate. Thus, for the Schick-Wolverton model, the aforementioned estimates may be obtained through equations 4 and 5.

$$\text{MTTF} = \sqrt{\frac{\pi}{2\emptyset[N-(i-1)]}} \qquad \text{(Eq. 4)}$$

$$R(t_i) = \exp\{-\emptyset[N-(i-1)]\ t_i^2/2\} \qquad \text{(Eq. 5)}$$

This particular model, as with many of the other software models, requires as inputs, the sequence of times between errors (i.e. t_1, t_2, t_3,.....,t_n).

Software error data has been and continues to be available only on a limited basis. However, this model was applied to two sets of data which are discussed below.

The first data set was extracted from a report published by Honeywell [27]. This small data set was in the required form of times between error occurrences. The following figure describes this set of error data.

Flight Number	Time Between Errors (t_i)	Cumulative Time
2	4.0	4.0
3	2.2	6.2
8	10.1	16.3
15	12.6	28.9
43	42.2	71.1

Fig. 19 Honeywell Flight Test Data

Knowing the t_i's, it was possible to solve equations 2 and 3 for $\hat{N}$ and $\hat{\phi}$ respectively. Doing so yields

$$\hat{N} = 4.0620532$$

$$\hat{\phi} = 0.017566$$

Now it becomes possible to determine how accurate this model is in predicting MTTF. As a sample calculation, suppose we select the point i=3, which corresponds to flight number 8. Substituting the previously calculated values for $\hat{N}$ and $\hat{\phi}$ into the hazard function (equation 1) and knowing $t_3 = 10.1$, gives:

$$Z(t_3) = 0.017566\ [4.0620532 - (3-1)]\ 10.1$$

$$Z(t_3) = 0.36584$$

To measure the accuracy of this model we apply equation 4 as follows:

$$MTTF = \sqrt{\frac{3.14159}{2(0.017566)\ [4.0620532 - (3-1)]}}$$

$$MTTF = 6.5853$$

This same procedure may be applied for any point i (= 1,2,3,4,5 in this case). The predicted values are then compared with the MTTF's in Figure 20. The basic Schick-Wolverton model was compared against other models requiring the same inputs (t_i). The basic Schick-Wolverton does not give as good a prediction as the basic Jelinski-Moranda, yet it has the same disadvantages as the

S-W,J-M AND GEOMETRIC-POISSON MODEL RESULTS
(HONEYWELL FLIGHT TEST DATA)

Actual MTTF	Basic Schick-Wolverton	Error	Basic Jelinski-Moranda	Error	Geometric	Error	Best Estimate
4.0 hours	4.69	17.2%	3.76	6.0%	2.75	31.2%	Jelinski-Moranda
2.2 hours	5.40	145.5%	4.87	121.4%	5.08	130.9%	Jelinski-Moranda
10.1 hours	6.58	34.9%	6.92	31.5%	9.40	6.9%	Geometric
12.6 hours	9.18	27.1%	11.94	5.2%	17.39	38.0%	Jelinski-Moranda
42.2 hours	37.96	10.0%	43.62	3.4%	32.15	23.8%	Jelinski-Moranda

FIG. 20

basic Jelinski-Moranda. That is, it does not allow for errors introduced during debugging. The Geometric does allow for this, and the basic Schick-Wolverton does not predict any better than the Geometric model. In fact, the mean percent error is slightly less for the Geometric. In addition, the basic Schick-Wolverton never gives the best estimate. Therefore, both the basic Jelinski-Moranda model and the Geometric model are superior to the basic Schick-Wolverton model as applied to this data set.

Dr. Paul Moranda, in [80], also applied this model to data from the "F-11D program". Not only was this data set larger than the Honeywell data, but it was also recorded in CPU time which is much more desirable than calendar time. Moranda states "there is usually a startup effect which is evident". With this particular data set erratic error behavior exists until the sixth day of testing. Only then does a decreasing failure rate become evident. Also, at this point, there are 67 errors remaining in the software. As most of the details are given in [80], it should suffice to state the following:

$$\hat{N} = 41.5$$

$$\hat{\phi} = 0.2192$$

Considering there are 67 known errors remaining, N = 41.5 indicates a 38% "error" in the prediction process. Figure 21 below, compares the results of three models as applied to the "F-11D program".

Model	Basic Schick-Wolverton	Basic Jelinski-Moranda	Geometric-Poisson
Est. # of Rem. Errors (Actual = 67)	41.5	63.4	62.73
Parameter Estimates	$\hat{\phi} = 0.2192$ $\hat{N} = 41.5$	$\hat{\phi} = 0.035$ $\hat{N} = 63.4$	$\hat{K} = 0.6756$ $\hat{\lambda} = 20.348$
"Error"	38.1%	5.4%	6.4%

Fig. 21 Comparison of Three Models Applied to F-11D Data

Again, as with the Honeywell data, the basic Schick-Wolverton model yields poorer results than the other models. Hence, based on these two applications we feel that the basic Schick-Wolverton will not yield acceptable results in our particular analysis.

1.6.13 Extended Schick-Wolverton Model

The extended Schick-Wolverton model, suggested by Myron Lipow of the TRW Defense and Space Systems Group, is very similar to the basic Schick-Wolverton model. The most notable change is that this extension allows for more than one error in a given debugging period. Thus, the required inputs are the number of errors per interval where each debugging interval is of equal length (time). Otherwise, the model assumptions remain the same as with the basic Schick-Wolverton model. The reader may refer back to the basic model for such information.

The hazard function in this case is of the form:

$$Z(t_i) = \emptyset\,[N-(n_{i-1})]\,t_i \qquad \text{(Eq. 1)}$$

where t_i = the $i^{\underline{th}}$ debugging interval (Def. 1)

$\emptyset$ = a proportionality constant (Def. 2)

N = the total number of initial errors in the system (Def. 3)

n_{i-1} = the cumulative number of errors encountered through the $(i-1)^{st}$ time interval (Def. 4)

As can be seen, the hazard rate proposed for this model is nearly identical to that given with the basic Schick-Wolverton model. Here n_{i-1} replaces i-1, since errors per interval are the inputs rather than the times between errors. The model parameters, N and $\emptyset$, are estimated through the utilization of the following equations:

$$\sum_{i=1}^{m} \frac{m_i}{N-n_{i-1}} = \sum_{i=1}^{m} \emptyset\, t_i^2 / 2 \qquad \text{(Eq. 2)}$$

Note: In reference [124], Capt. Sukert omitted the $\emptyset$ in equation 2.

$$\frac{n}{\emptyset} = \sum_{i=1}^{m} [N-n_{i-1}]\, t_i^2 / 2 \qquad \text{(Eq. 3)}$$

Now, if equation 3 is solved for $\emptyset$ and substituted into equation 2, we obtain:

$$n = \left\{ \sum_{i=1}^{m} \frac{m_i}{N-n_{i-1}} \right\} \left\{ N - \frac{\sum_{i=1}^{m} n_{i-1}\, t_i^2 / 2}{\sum_{i=1}^{m} t_i^2 / 2} \right\} \qquad \text{(Eq. 4)}$$

where

m = the total number of intervals (Def. 5)

m_i = the number of errors found in the $i^{\underline{th}}$ time interval (Def. 6)

n = the total number of errors found to date (Def. 7)

Then, similar to the basic Schick-Wolverton model, MTTF's and $R(t_i)$ may be calculated from the following expressions:

$$MTTF = \sqrt{\frac{\pi}{2\,\emptyset\,(N-n_{i-1})}} \qquad \text{(Eq. 5)}$$

$$R(t_i) = \exp\left(-\emptyset\,(N-n_{i-1})\, t_i^2 / 2\right) \qquad \text{(Eq. 6)}$$

As previously mentioned, m_i, the number of errors per interval (i.e. $m_1, m_2, m_3, \ldots, m_m$) is a required input in order to apply this extended model.

The acquisition of two sets of data in the proper format made it possible to apply the extended Schick-Wolverton model. First, a small set of data, from Dickson et.al [30], was tested. The weaknesses associated with this data are that the set is quite small; however, a second limiting factor is that the intervals were in calendar time as opposed to CPU time.

Below, Figure 22, is a listing of the errors per month and the cumulative number of errors to date found in reference [30].

Month	Errors/Month (m_i)	Cumulative Number of Errors Thru $(i-1)^{st}$ Interval (n_{i-1})
1	520	0
2	430	520
3	300	950
4	170	1250
5	120	1420
6	60	1540
7	40	1600

Fig. 22 Dickson et.al. Software Error Data

$$n = \sum_{i=1}^{m} m_i = 1640, \; m = 7$$

To estimate the model parameters, equation 4 must be solved. Doing so will yield a value for N. Using the m_i and n_{i-1} given in Figure 22 produces

$$\hat{N} = 1738.6$$

Now, applying equation 2 or 3 we find

$$\hat{\phi} = 0.6708571$$

Note that 1640 errors have been found to date and the model predicts approximately 1739 errors overall. We feel that this is a fairly reasonable estimate.

However, one notable shortcoming of this model becomes evident when the hazard function is applied. The sample calculations below, Figure 23, and Figure 24 will amplify this deficiency.

Suppose we wish to estimate the number of errors in interval (month) 4, which has been selected arbitrarily. Substituting the known values into equation 1 (the hazard function) will give

$$Z(t_4) = 0.6708571 \; (1738.6 - 1250)1$$

$$Z(t_4) = 327.78$$

COMPARISON OF THE EXTENDED J-M, GEOMETRIC-POISSON AND EXTENDED S-W MODELS (DICKSON ET. AL. DATA)

Actual Number of Errors	Geometric-Poisson	Error	Extended J-M	Error	Extended S-W	Error	Best Estimate
520	582.00	11.9%	583.09	12.1%	1166.35	124%	Geometric-Poisson
430	387.46	9.9%	408.70	5.0%	817.51	90%	Extended J-M
300	257.94	14.0%	264.48	11.8%	529.04	76.3%	Extended J-M
170	171.72	1.0%	163.87	3.6%	327.78	92.8%	Geometric-Poisson
120	114.32	4.7%	106.85	11.0%	213.74	78.1%	Geometric-Poisson
60	76.11	26.8%	66.61	11.0%	133.23	122%	Extended J-M
40	50.67	26.7%	46.48	16.2%	92.98	132%	Extended J-M

FIG. 23

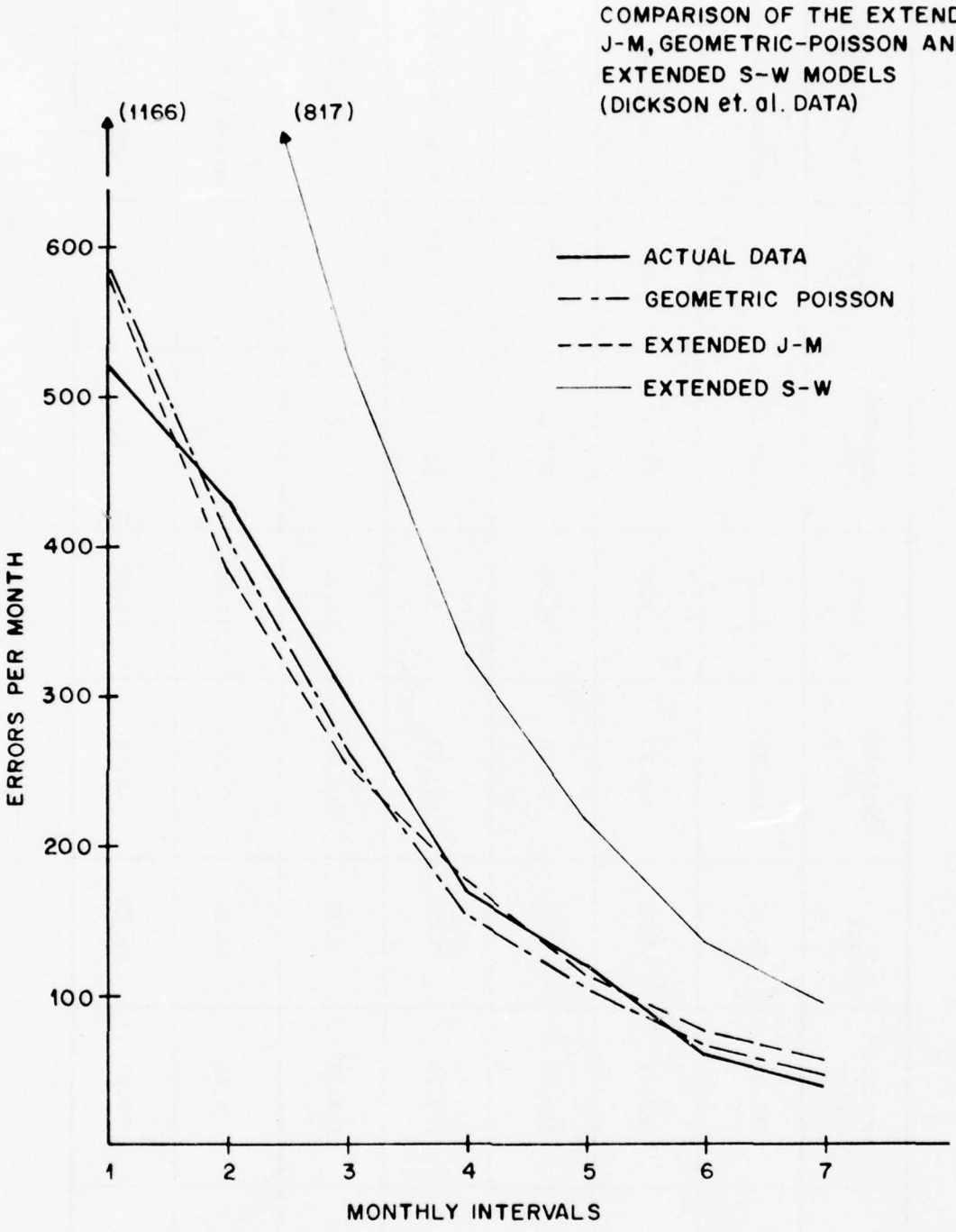

COMPARISON OF THE EXTENDED
J-M, GEOMETRIC-POISSON AND
EXTENDED S-W MODELS
(DICKSON et. al. DATA)
(1166)
(817)
ACTUAL DATA
GEOMETRIC POISSON
EXTENDED J-M
EXTENDED S-W
ERRORS PER MONTH
600
500
400
300
200
100
1
2
3
4
5
6
7
MONTHLY INTERVALS

FIG. 24

Comparing the predicted error count (327.78) with the actual value (170) shows the inaccuracy of the monthly predictions. The 93% variation is extremely large and Figure 23 gives the remaining predictions and the respective "error" (variation).

One puzzling point is that the model predicted 1738.6 total errors in the system, however, summing the monthly estimates gives a total of 3280.63. This alone causes us to suspect the adequacy of the model.

Both Figure 23 and Figure 24 clearly show that the Geometric-Poisson and the extended Jelinski-Moranda models are much more accurate than the extended Schick-Wolverton in the monthly prediction process.

The second set of data that this model was applied to was obtained from reference [134]. In this case, errors were given in CPU time (seconds) and 10 second intervals were used for t_i which required interpolation to find the number of errors per interval. Figure 25 below lists the number of errors per interval as well as the cumulative number of errors discovered.

Interval	Errors/Interval (m_i)	Cumulative Number of Errors Thru $(i-1)^{st}$ Interval (n_{i-1})
0-10 sec.	19.53	0
10-20 sec.	12.83	19.53
20-30 sec.	10.93	32.36
30-40 sec.	7.52	43.29
40-50 sec.	4.58	50.81
50-60 sec.	1.37	55.39

Fig. 25 F-11D Error Data (Errors Per Interval)

$$n = \sum_{i=1}^{m} m_i = 56.76, \quad m=6$$

Moranda, in [80], mentions that the errors behave in an erratic manner early (up through 1/18) and attributes this to the startup effect of the program. Thus, the data used in the model application begins with day 1/19. The above table takes the aforementioned statements into account.

Now, once again estimates for N and $\emptyset$ must be obtained. Utilizing the same procedure as was used with the Dickson et.al. data we find

$$\hat{N} = 62.2$$

and

$$\hat{\emptyset} = 0.6607$$

To date 56.76 errors have been found and the extended Schick-Wolverton model predicted 62.2. Again, this estimate appears to be fairly accurate. Figures 26 and 27 were derived upon application of the hazard function for this model. For example, if we want to find the estimated number of errors in the second interval, we find (by equation 1)

$$Z(t_2) = 0.6607(62.2 - 19.53)\ 1$$

$$Z(t_2) = 28.19$$

Figure 26 indicates that this estimate is very inaccurate when compared with 12.83 (the actual value). As previously mentioned, the extended Schick-Wolverton predicted 62.2 errors (after day 1/18); however, summing the six intervals gives 113.53 errors. We have no explanation as to why this inconsistency occurs. Also, as with the other data set (Dickson et.al.), again, the extended Schick-Wolverton model was inferior to both the Geometric-Poisson and the extended Jelinski-Moranda models. Our conclusion is that this particular model does not yield accurate results when predicting the number of errors per interval. Thus, this model has been eliminated from further consideration concerning our software analysis study.

1.6.14 Modified Schick-Wolverton Model

The modified Schick-Wolverton model, a slight variation of the extended Schick-Wolverton model, was suggested by Lipow and implemented by Sukert [123,124].

EXTENDED S-W, EXTENDED J-M AND GEOMETRIC-POISSON MODEL RESULTS
(F-11D DATA)

Actual Number of Errors	Geometric-Poisson	Error	Extended J-M	Error	Extended S-W	Error	Best Estimate
19.53	20.35	4.2%	20.55	5.2%	41.10	110%	Geometric-Poisson
12.83	13.75	7.2%	14.10	9.9%	28.19	120%	Geometric-Poisson
10.93	9.28	15.1%	9.86	9.8%	19.72	80%	Extended J-M
7.52	6.27	16.6%	6.25	16.9%	12.50	66%	Geometric-Poisson
4.58	4.23	7.6%	3.76	17.9%	7.52	64%	Geometric-Poisson
1.37	2.86	108%	2.25	64.2%	4.50	228%	Extended J-M

FIG. 26

EXTENDED S-W, EXTENDED J-M AND GEOMETRIC-POISSON MODEL RESULTS
(F-11D DATA)

Actual Number of Errors	Geometric-Poisson	Error	Extended J-M	Error	Extended S-W	Error	Best Estimate
19.53	20.35	4.2%	20.55	5.2%	41.10	110%	Geometric-Poisson
12.83	13.75	7.2%	14.10	9.9%	28.19	120%	Geometric-Poisson
10.93	9.28	15.1%	9.86	9.8%	19.72	80%	Extended J-M
7.52	6.27	16.6%	6.25	16.9%	12.50	66%	Geometric-Poisson
4.58	4.23	7.6%	3.76	17.9%	7.52	64%	Geometric-Poisson
1.37	2.86	108%	2.25	64.2%	4.50	228%	Extended J-M

FIG. 26

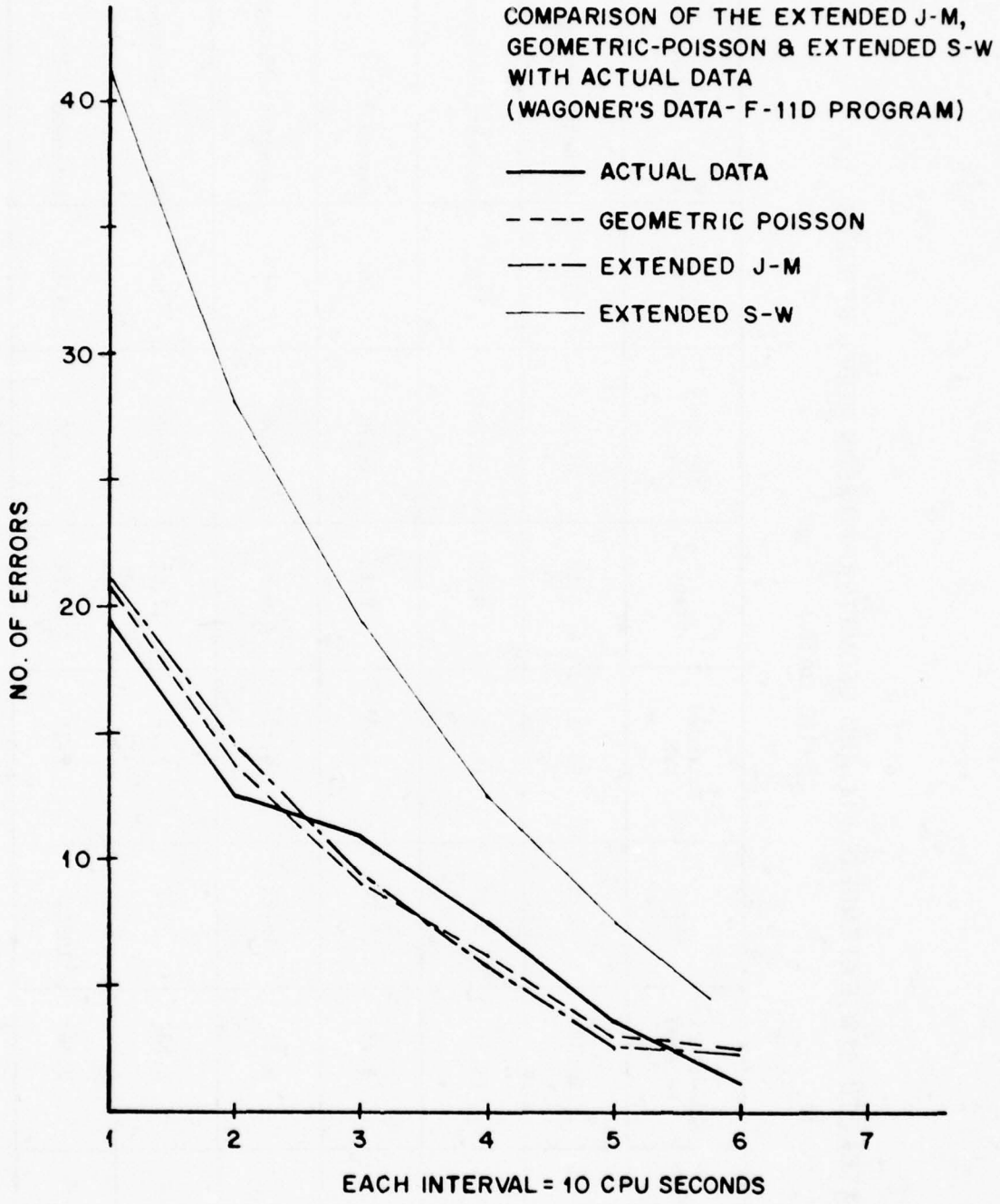

COMPARISON OF THE EXTENDED J-M,
GEOMETRIC-POISSON & EXTENDED S-W
WITH ACTUAL DATA
(WAGONER'S DATA- F-11D PROGRAM)
ACTUAL DATA
GEOMETRIC POISSON
EXTENDED J-M
EXTENDED S-W
NO. OF ERRORS
40
30
20
10
1
2
3
4
5
6
7
EACH INTERVAL = 10 CPU SECONDS

FIG. 27

The assumptions of the modified Schick-Wolverton model are listed and discussed below.

(1) There is a fixed number of errors in the program.

(2) No new errors are added during debugging.

(3) The error rate is constant during a time interval and is proportional to the number of errors remaining following the $(i-1)^{st}$ time interval and the total debugging time previously spent, including an "averaged" error search time during the current time interval.

(4) Each error discovered is immediately removed.

The major difference with the modified version is assumption 3. It is assumed that the error discovery rate is constant during a time interval and is proportional to the number of errors remaining following the $(i-1)^{st}$ time interval and the total time previously spent in testing, including an "averaged" error search time during the current time interval t_i (Figure 28). We strongly question this assumption, which implies that the error detection rate increases with the time spent in searching for the i^{th} error.

Furthermore, we question the relationship which indicates that the failure rate increases as a function of total time spent in debugging. As can be seen in Figure 28, this could result in an increasing failure rate when the number of detected failures is relatively low. It should be noted however that as the number of detected errors becomes large, the error rate between intervals tends to decrease, and within intervals, the rate of increase of the failure rate with time becomes smaller, approaching a constant rate within the interval.

However, the hazard function for the modified Schick-Wolverton model is given as:

$$Z(t_i) = \emptyset\ (N-n_{i-1})\ (T_{i-1} + t_i/2) \qquad \text{(Eq. 1)}$$

Note: The formulation of the modified Schick-Wolverton model is described by Sukert in references [123] and [124]. In reference [123], $\emptyset$, the proportionality

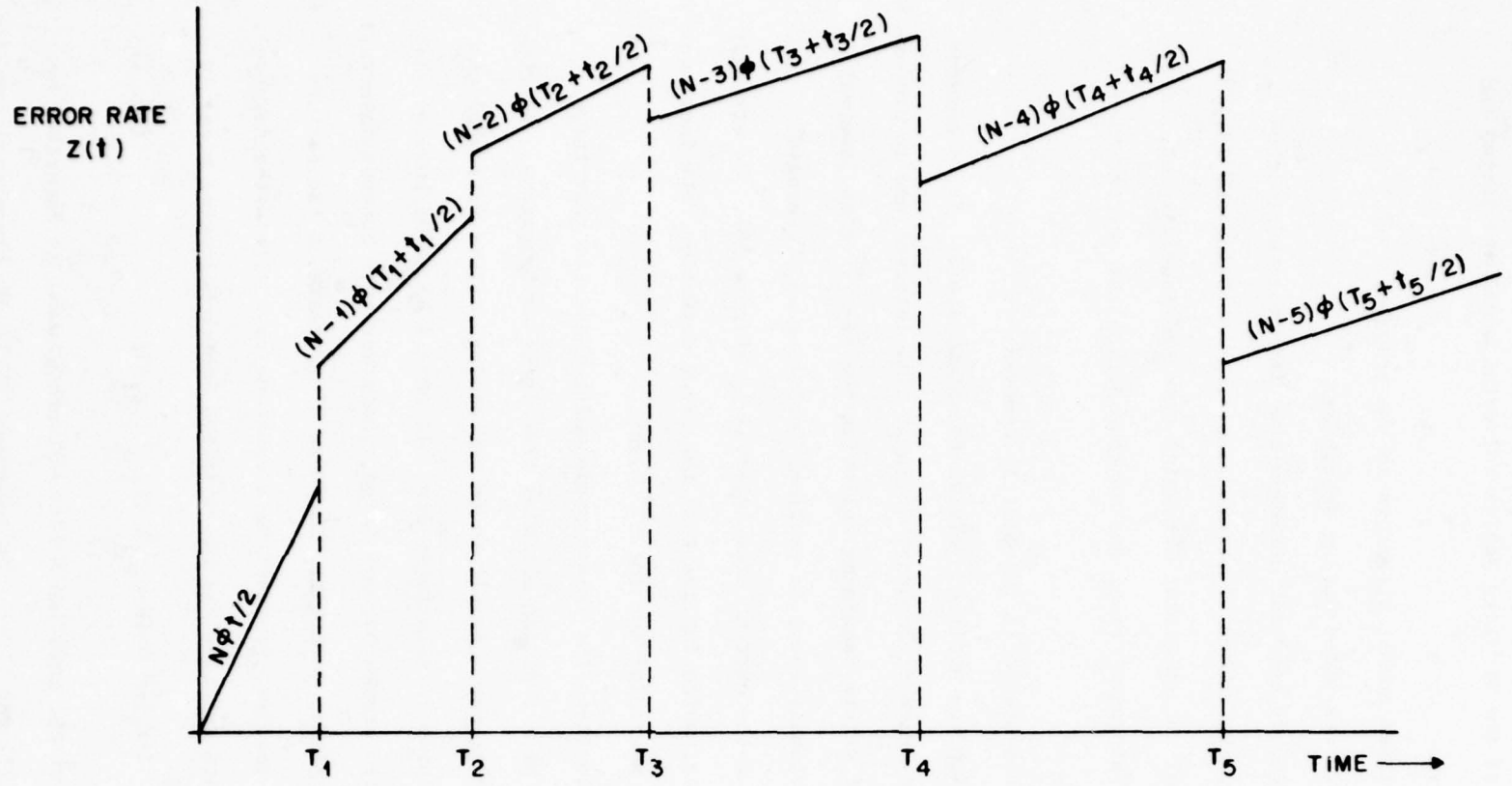


FIG. 28- MODIFIED SCHICK WOLVERTON MODEL

FAILURE RATE PROPORTIONAL TO THE NUMBER OF REMAINING ERRORS, THE TOTAL TIME SPENT IN DEBUGGING TO DATE, AND THE "AVERAGE" TIME SINCE THE DETECTION OF THE LAST ERROR.

constant, was omitted from the above hazard function. However, reference [124] does include $\emptyset$ in the aforementioned hazard function for this model. Thus, it is our conclusion that the deletion of $\emptyset$ from the hazard function in [123] was merely a typographical error.

Sukert also stated that the hazard rate between errors in this modified version of the model is constant, which is inconsistent with any plausible interpretation of the hazard function for this model.

In any event, the notation for the hazard function is defined below.

t_i = the i^{th} debugging interval (Def. 1)

$\emptyset$ = a proportionality constant (Def. 2)

N = the total number of initial errors in the system (Def. 3)

n_{i-1} = the cumulative number of errors observed up through the $(i-1)^{st}$ interval (Def. 4)

T_{i-1} = the cumulative debugging time through the $(i-1)^{st}$ interval (Def. 5)

Again, N and $\emptyset$ are the model parameters and may be estimated by the following equations:

$$\sum_{i=1}^{m} \frac{m_i}{N-n_{i-1}} = \sum_{i=1}^{m} \emptyset\, t_i\, (T_{i-1} + t_i/2) \qquad \text{(Eq. 2)}$$

Note: $\emptyset$ was omitted from the above equation by Capt. Sukert in reference [124].

$$\frac{n}{\emptyset} = \sum_{i=1}^{m} (N-n_{i-1})\, t_i\, (T_{i-1} + t_i/2) \qquad \text{(Eq. 3)}$$

Now, solving equation 3 for $\emptyset$ and substituting back into equation 2, yields

$$n = \left[\sum_{i=1}^{m} \frac{m_i}{N-n_{i-1}}\right] \left[N - \frac{\sum_{i=1}^{m} n_{i-1}\, t_i^2/2}{\sum_{i=1}^{m} t_i^2/2}\right] \qquad \text{(Eq. 4)}$$

where m = the total number of intervals (Def. 5)

m_i = the number of errors found in the i^{th} time interval (Def. 6)

n = the total number of errors found to date (Def. 7)

Next, MTTF and $R(t_i)$ may be computed using the following formulas:

$$MTTF = \int_0^\infty R(t_i)\, dt_i \qquad \text{(Eq. 5)}$$

$$R(t_i) = \exp\left(- \emptyset\, (N-n)\, (T_{i-1}\, t_i + t_i^2 /4) \right) \qquad \text{(Eq. 6)}$$

Since the modified Schick-Wolverton model requires the same inputs as the extended Schick-Wolverton model (m_i) we will apply this model to the same two data sets.

First, the Dickson et.al. data set is given in Figure 22 (under extended Schick-Wolverton). Using equations 2-4 we may calculate $\hat{N}$ and $\hat{\emptyset}$. The modified Schick-Wolverton model yields the following estimates:

$$\hat{N} = 1613.85$$

and

$$\hat{\emptyset} = 0.2430$$

Immediately, it is evident that $\hat{N}$ (1613.85) is not a very good estimate since 1640 errors have already been found. Again, as with the extended Schick-Wolverton, we find that summing the monthly predictions yields a value in excess of $\hat{N}$ (the total number of predicted errors in the system). Figure 29 and Figure 30 show the results when the modified model is applied to the previously discussed data set.

Again, a sample calculation is presented to show the procedure for estimating the number of errors in an interval. For month 7 equation 1 gives

$$Z(t_7) = 0.2430\ (1613.85 - 1600)\ (6 + 1/2)$$

$$Z(t_7) = 21.878$$

Figure 29 gives the predicted values for all intervals and compares them with the actual values. Also, the extended Jelinski-Moranda and Geometric-Poisson models' results are compared with those obtained from the modified Schick-Wolverton model. As can be seen, the results of the modified model are not encouraging.

EXTENDED J-M, MODIFIED S-W AND GEOMETRIC-POISSON MODEL RESULTS (DICKSON ET AL. DATA)

Actual Number of Errors Per Interval	Geometric-Poisson	Error	Extended J-M	Error	Modified S-W	Error	Best Estimate
520	582.00	11.9%	583.09	12.1%	196.10	66.2%	Geometric-Poisson
430	387.46	9.9%	408.70	5.0%	398.75	7.3%	Extended J-M
300	257.94	14.0%	264.48	11.8%	403.33	34.4%	Extended J-M
170	171.72	1.0%	163.87	3.6%	309.48	82.0%	Geometric-Poisson
120	114.32	4.7%	106.85	11.0%	211.99	76.7%	Geometric-Poisson
60	76.11	26.8%	66.61	11.0%	98.71	64.5%	Extended J-M
40	50.67	26.7%	46.48	16.2%	21.88	45.3%	Extended J-M

FIG. 29

COMPARISON OF THE EXTENDED J-M, GEOMETRIC-POISSON AND MODIFIED S-W MODELS (DICKSON et. al. DATA)

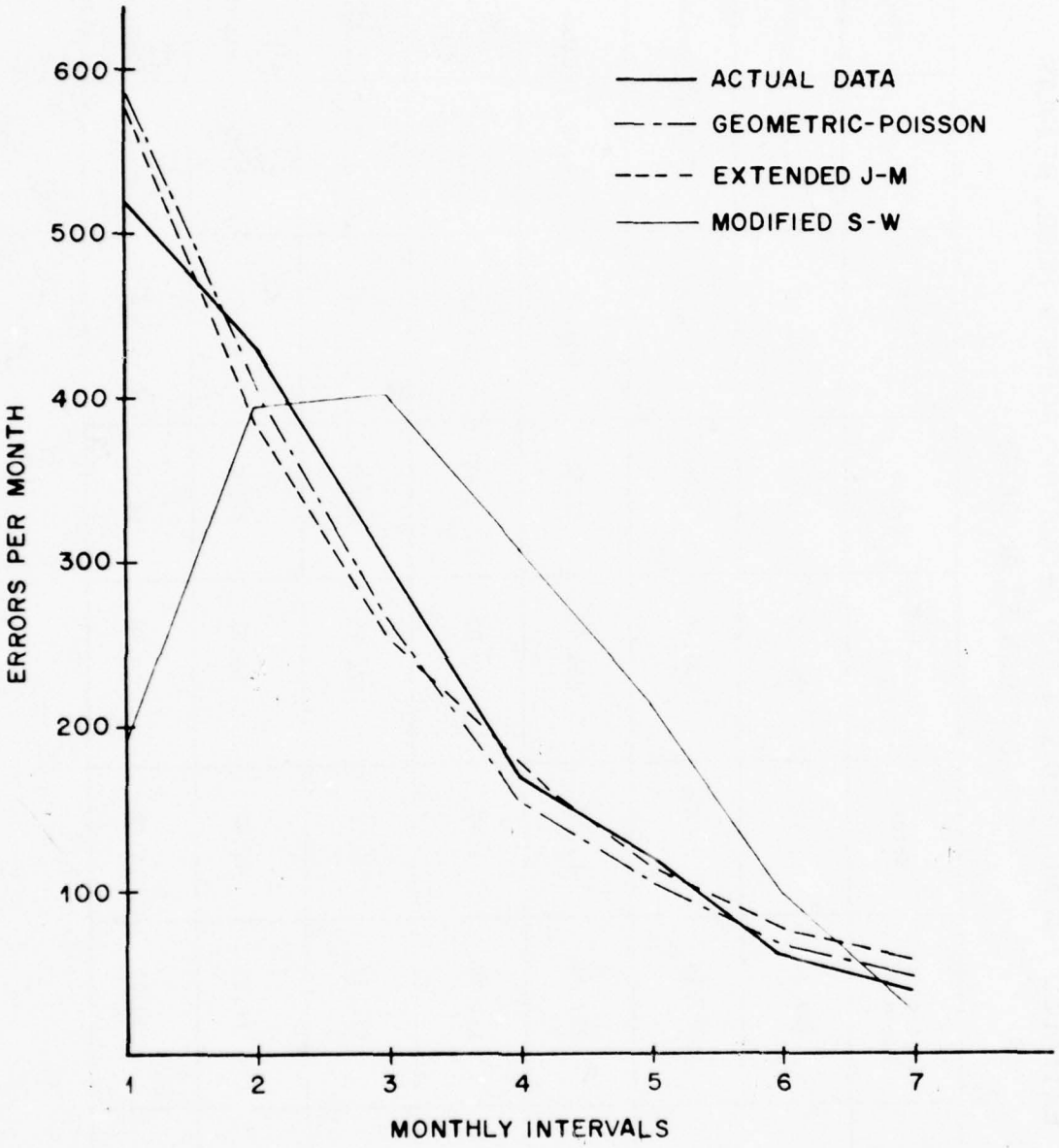


FIG. 30

The F-11D data, given in reference [134] was also applied and is considered to be good data since CPU seconds was the time scale as opposed to calendar time. Figure 25 (also given in section 1.6.13) gives the necessary inputs for this model. The equations for parameter estimation (Equations 2-4) were used to obtain

$$\hat{N} = 56.0244$$

$$\hat{\phi} = 0.26593$$

It should be noted that 45.76 errors have already been found and this model predicts a total of 56.0244. This estimate is not as good as it initially appears, and Figures 31 and 32 are obtained when we apply the hazard function to all of the intervals. Once again, this model is much less accurate than either the Geometric-Poisson or the extended Jelinski-Moranda models' estimates. Our conclusion here, as with the extended Schick-Wolverton model, is that this particular model does not appear to model the error process too well; hence we see little reason in further pursuing it.

1.6.15 LaPadula Reliability Growth Model

This model was developed in [61] and is an outgrowth of a hardware reliability growth model. It makes the following assumptions:

(1) A test sequence is conducted in N states, where a stage terminates whenever a change (of any kind) is made to the program.

(2) The change can be either a correction to or a corruption of the program.

(3) The number of tests per stage of testing is not fixed.

(4) The number of stages in the test sequence is not fixed.

(5) At any particular stage of testing, there exists an upper bound on the reliability of a piece of software, and this upper bound may very well be significantly less than one.

EXTENDED J-M, MODIFIED S-W AND GEOMETRIC-POISSON MODEL RESULTS (F-11D DATA)

Actual Number of Errors Per Interval	Geometric-Poisson	Error	Extended J-M	Error	Modified S-W	Error	Best Estimate
19.53	20.35	4.2%	20.55	5.2%	7.45	61.8%	Geometric-Poisson
12.83	13.75	7.2%	14.10	9.9%	14.56	13.5%	Geometric-Poisson
10.93	9.28	15.1%	9.86	9.8%	15.73	43.9%	Extended J-M
7.52	6.27	16.6%	6.25	16.9%	11.85	57.6%	Geometric-Poisson
4.58	4.23	7.6%	3.76	17.9%	6.24	36.2%	Geometric-Poisson
1.37	2.86	108%	2.25	64.2%	0.93	32.1%	Modified S-W

FIG. 31

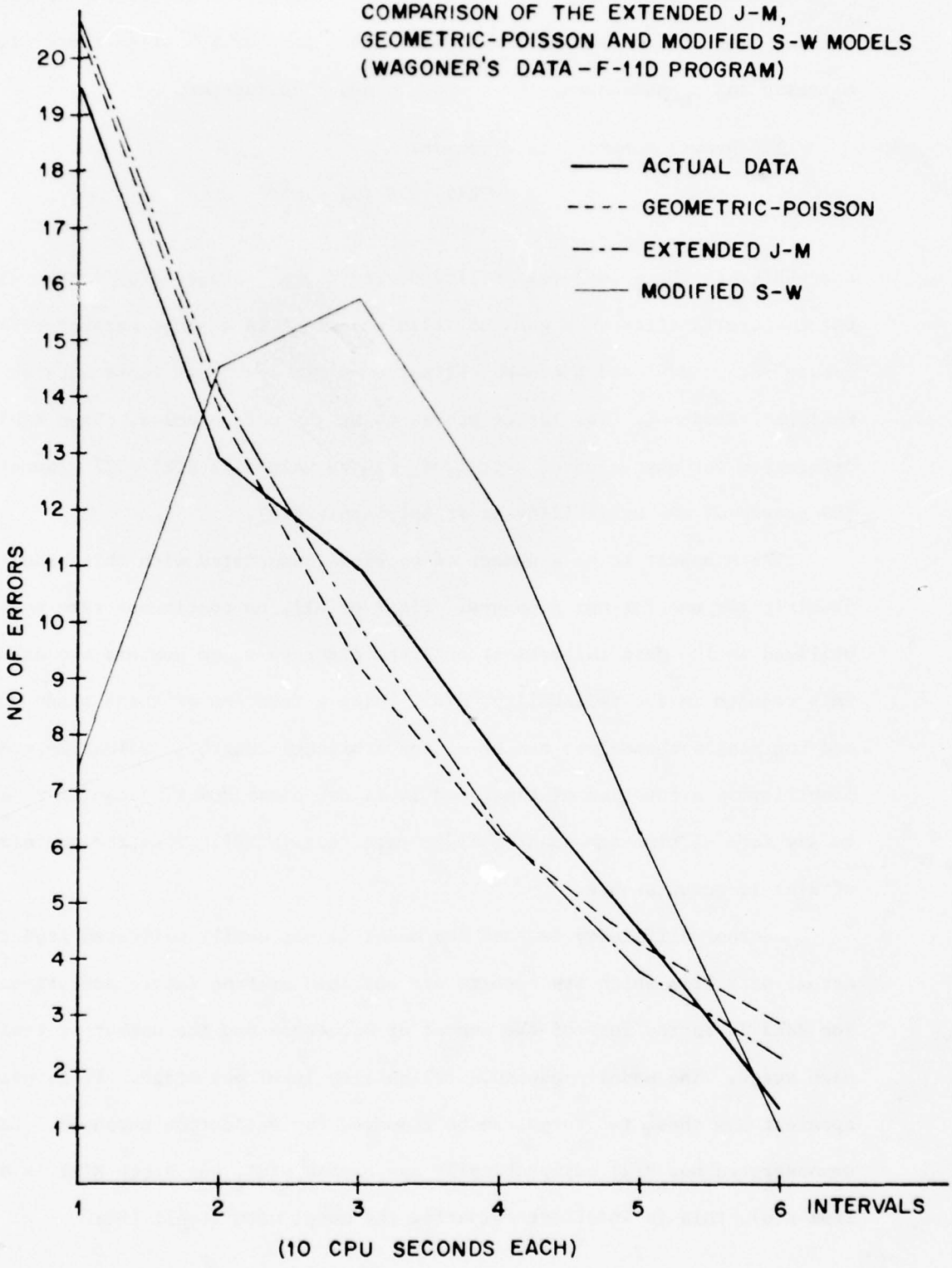

COMPARISON OF THE EXTENDED J-M,
GEOMETRIC-POISSON AND MODIFIED S-W MODELS
(WAGONER'S DATA - F-11D PROGRAM)
ACTUAL DATA
GEOMETRIC-POISSON
EXTENDED J-M
MODIFIED S-W
NO. OF ERRORS
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1
2
3
4
5
6
INTERVALS
(10 CPU SECONDS EACH)

FIG. 32

The only data recorded is whether the program was successful or failing in each test during each testing state. Thus for the k^{th} stage there will be n_k tests and s_k successes, for $1 \leq k \leq n$ and k an integer.

The growth function is given as:

$$R(k) = R(u) - \frac{A}{k}$$

where R(k) is the actual reliability during the k^{th} stage, R(u) is the ultimate value of reliability as k goes to infinity, and A is a shape parameter for the growth rate. R(u) and A can be estimated by the method of least squares or by maximum likelihood. The latter proves to be the more precise. Then R(k) can be determined for any value of k desired. These values of R(k) will demonstrate the growth of the reliability as it approaches R(u).

There appear to be a number of problems associated with this model which prohibit its use for our purposes. First of all, no continuous time scale is utilized in the data collection; only the discrete stage numbers are used. This results in the reliability, R(k), being a function of these stage numbers, and the stages themselves can be of any arbitrary length. Ordinarily, reliability is a function of time, and it is not clear how R(k) can be related to any form of time-dependent failure rate (e.g. MTTF). Thus the usefulness of R(k) is questionable.

Another difficulty is that the model is not easily validated from the actual data from which its results are obtained or from future operational data. The data is in the form of the number of successes and the number of tests in each stage. The model produces a reliability level per stage. It is not apparent how these two forms can be compared for validation purposes. LaPadula demonstrates how R(k) asymptotically approaches R(u), but since R(k) is obtained from R(u), this is in effect comparing the model with itself [61].

In most software packages there are a number of failures at the beginning of testing. If R(u) is estimated from this data it may be unreasonably low. The model does not indicate how long data should be collected to lessen the impact of these early failures and to arrive at a "reasonable" R(u). Even if such a stopping rule existed, least squares and maximum likelihood techniques for estimation are hardly infallible, and little confidence could be placed in R(u), especially if the software is a part of some critical, real time control system (e.g. air traffic control). Associated with this problem is the fact that as $k \to \infty$ there will be an infinite number of changes to the software, and it may have changed so drastically that it may no longer be the same software.

Because of the above problems we do not feel that this model will be useful in our effort.

1.7 APPLICATION OF PROMISING MODELS TO DATA

After reviewing the assumptions and formulations of the models described in Sections 1.5 and 1.6 we wished to make a more extensive comparison of the best models when applied to a number of data sets. The available literature contains or describes some 25 data sets. Most of these, however, are not completely suitable. Many are quite small, with five to eight data points, others are only presented graphically, and not tabulated. After a review and comparison of the available data sets we selected four for further study. These were Sukert's data set one [124], Musa's example one [83], Baker's report of IBM development experience to Rome Air Development Center (RADC) [8], and a set of data gathered from Damman's report on the flight test of a digital flight control system [27].

Four of the models were selected as being "best". They were the Jelinski-Moranda exponential model, the extended Jelinski-Moranda exponential model, the Geometric de-eutrophication model and the Schneidewind method 3.

This group of four models was considered broad enough to meet the limitations of most data sets. Both the Jelinski-Moranda model and the Geometric model require data collected about the sequence of times between errors, while the extended Jelinski-Moranda and Schneidewind models require data giving the number of errors in some uniform time period (e.g. errors per day).

The Jelinski-Moranda and extended Jelinski-Moranda are typical of a family of exponential models which also includes the Shooman, Miyamoto and Musa models. They are distinguished by a hazard/correction rate propotional to the number of errors remaining in the program and the assumption of a fixed initial number of errors in the program with no new errors being introduced during debugging. The Geometric model and Schneidewind's method 3 are representative of another family of models which assume a hazard/correction rate proportional to the rate of the previous interval or error. This allows for new errors introduced during the debugging process, so long as the number of new errors created is proportional to the number of old errors corrected. Other models in this family are Schneidewind's methods 1 and 2 and the Geometric-Poisson.

In the absence of data on the long term performance of specific software, much of the data provided by a reliability model can be of little use. In this circumstance such information as total initial errors, number of errors remaining and the like is only of academic interest. After all, when the only data available is time between errors, all that can be validated from a model is predicted time between errors. Additionally, the only values available from all the models are either predicted and estimated past time between errors, or predicted and estimated past errors per period. While many of the exponential family models provide an estimate of total initial errors most other models do not. Several papers have presented results using cumulative errors detected. The use of either this or some calculation of errors remaining introduces an undesirable smoothing effect, due to the reduced relative error. Specifically, the impact of this smoothing effect can be seen in Figures 33 and 34. Figure 33

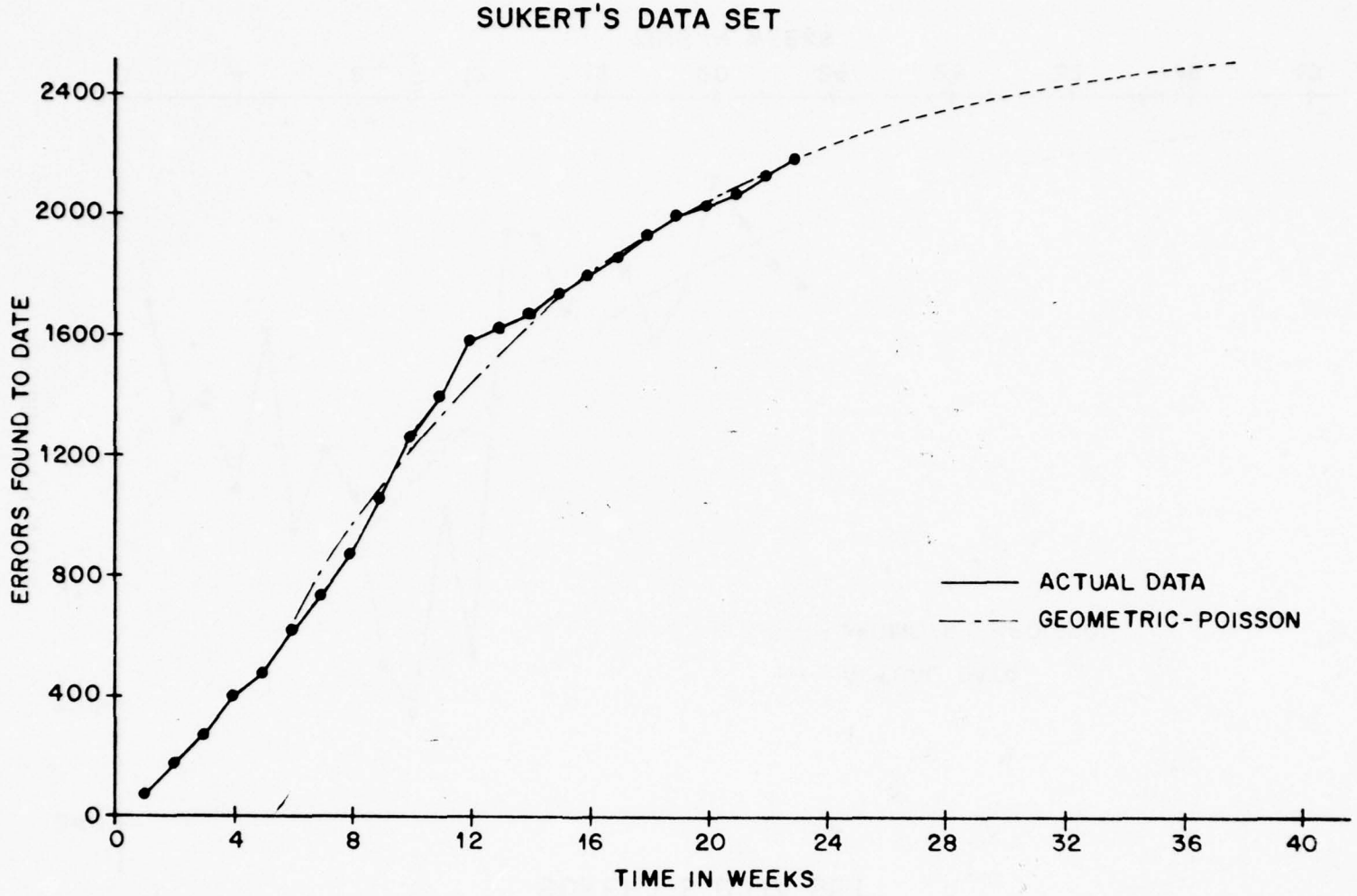

SUKERT'S DATA SET
ERRORS FOUND TO DATE
2400
2000
1600
1200
800
400
0
0
4
8
12
16
20
24
28
32
36
40
TIME IN WEEKS
ACTUAL DATA
GEOMETRIC-POISSON

FIG. 33

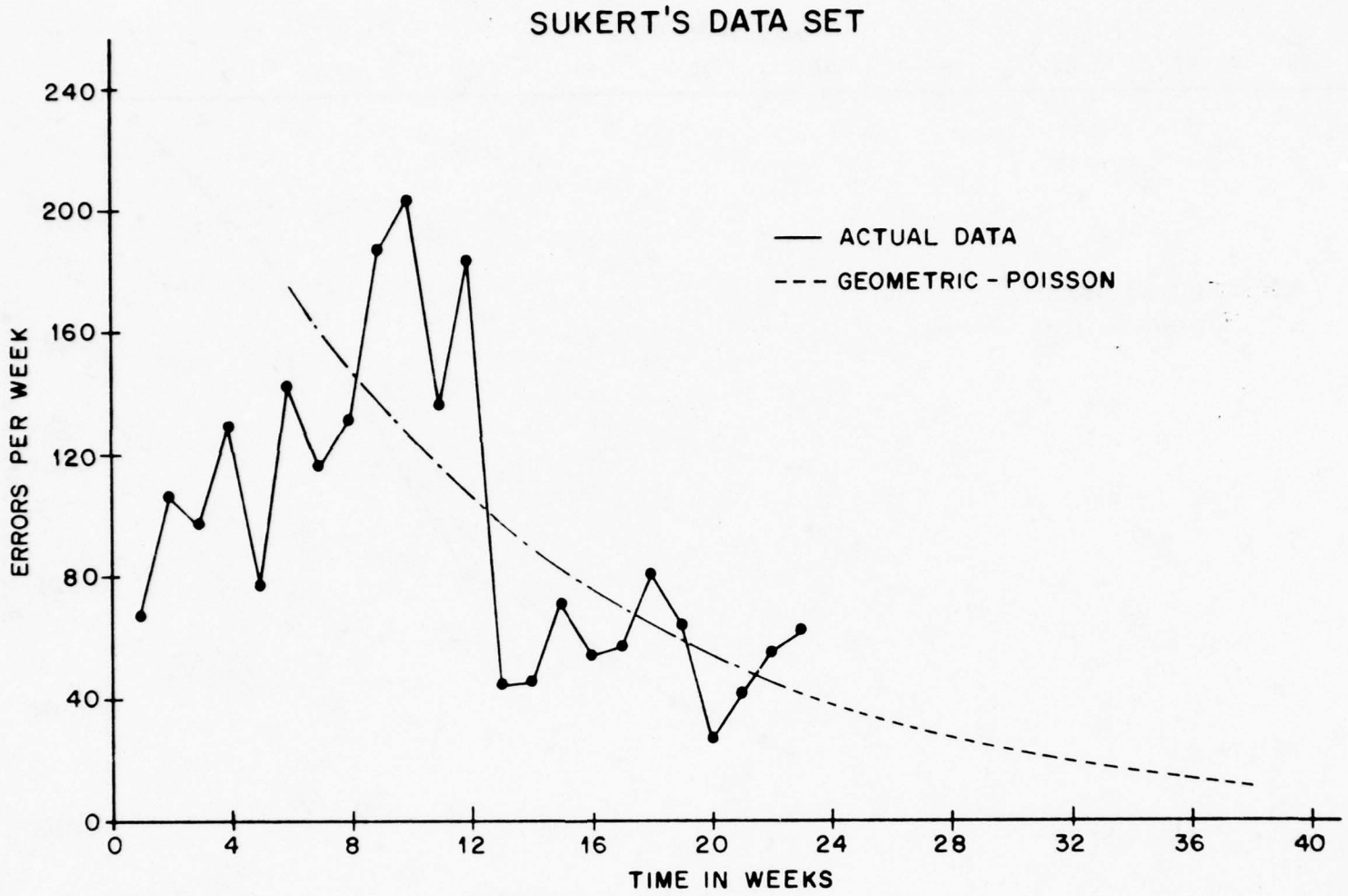

SUKERT'S DATA SET
ACTUAL DATA
GEOMETRIC - POISSON
ERRORS PER WEEK
TIME IN WEEKS
0
40
80
120
160
200
240
0
4
8
12
16
20
24
28
32
36
40

FIG. 34

is a plot of cumulative errors found from Sukert's data (described below) and the Geometric-Poisson model fitted to the last 18 weeks. Figure 34 plots the error data on a week by week basis. The use of errors remaining in the evaluation of model performance introduces another problem as well. This is the difficulty of estimating or determining what the actual initial number of errors in the software was. Several authors have assumed that all errors in the software under study had been detected at the time the study began. For data like that of Figure 35 such an assumption is clearly unwarranted. It should be noted that at least one author has even declined to count those errors discovered during his study.

1.7.1 Sukert's Data

A.N. Sukert reports in [124] and summarizes in [123] an investigation of several software reliability models. He compared the performance of the extended Jelinski-Moranda, Schick-Wolverton, extended Schick-Wolverton, and Geometric-Poisson models. His data is a sequence of 2191 errors grouped by day of occurrence over a 165-day period. He describes the data as being from an Air Force command and control system, and it is reproduced in Figure 35. Throughout his study Sukert evaluates the models based on their ability to predict the number of remaining errors. As described above, the uncertainty in determining what the actual number of remaining errors really is, together with the fact that models of the geometric family have no upper bound on the number of remaining errors, argues against that approach for this study.

We feel that a major requirement of a study of this type is the use of all available data, exactly as given. Many of the papers presenting models use published data sets, but trim off or "bank" errors from the beginning and end of testing with little or no explanation or justification. While this can

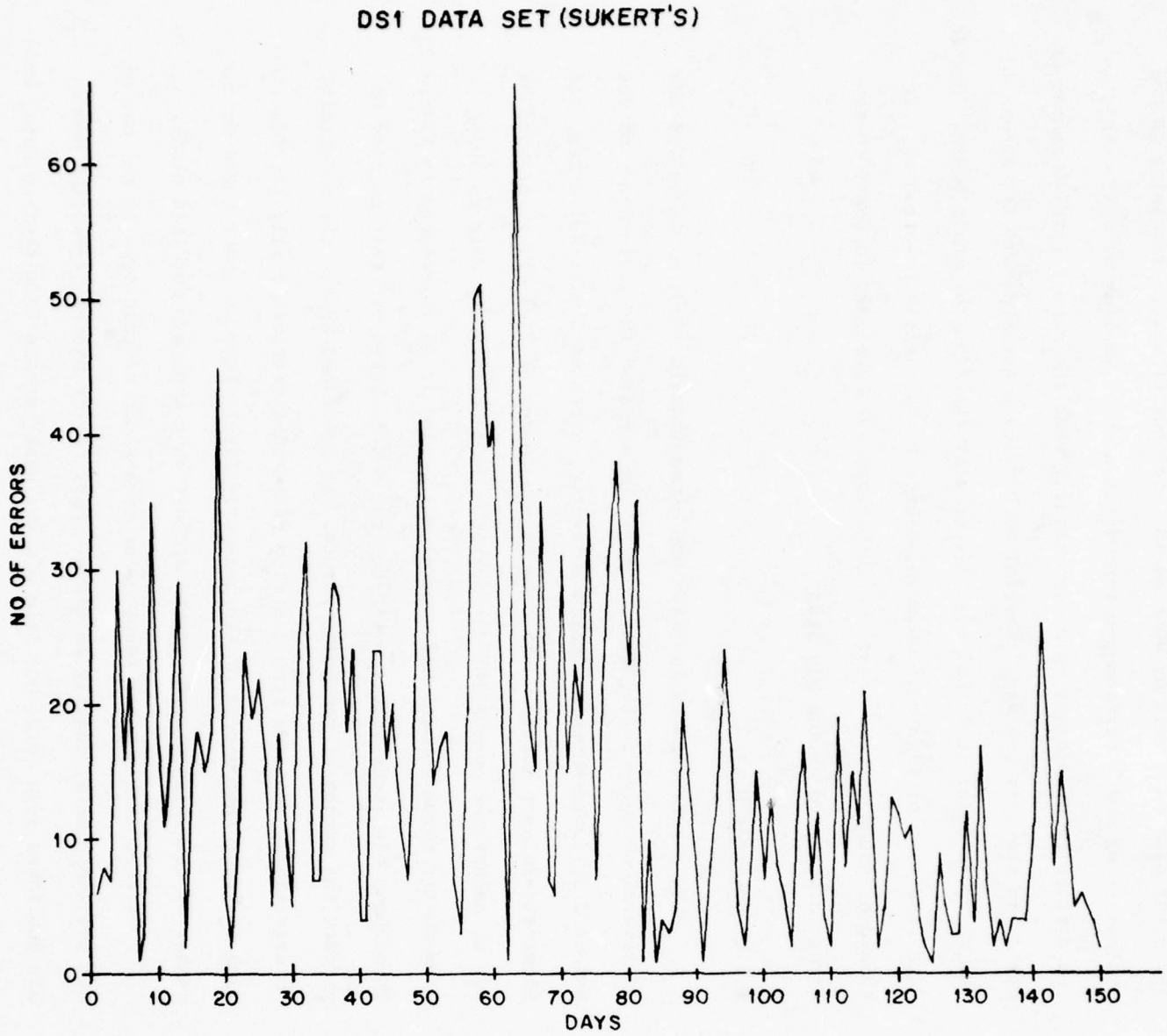

DS1 DATA SET (SUKERT'S)
NO OF ERRORS
0
10
20
30
40
50
60
DAYS
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150

FIG. 35

certainly improve a model's performance, it should require very good justification. This particular data set shows a strong seven day cycle, due perhaps to reduced testing on weekends. In order to get a more consistent time scale with respect to debugging effort and error exposure, we chose to group the data into 23 weekly intervals. This data, as tabulated in Figure 36 and plotted in Figure 37, was analyzed with the extended Jelinski-Moranda exponential model and Schneidewind's method 3.

The results using the two models are also listed in Figure 36. As can be seen, these two models give very similar results, differing by nine errors per week at most. They do appear to give equally good fit. The extended Jelinski-Moranda is less smooth due to its use of actual errors in its hazard function. Figure 38 plots the relationship between the extended Jelinski-Moranda model and the data. Two differences between the models did become apparent when we began solving them for smaller subsets of the data. As seen in Figure 39, the extended Jelinski-Moranda model is somewhat inconsistent. Fairly small changes in the data give a significant change in the model's shape and prediction. Also, the effect of the assumption of a finite number of errors is quite apparent since the extended Jelinski-Moranda model reports no errors left five weeks before the end of testing.

1.7.2 IBM Data

W.F. Baker of IBM, in a report to Rome Air Development Center, describes and analyzes software error reports from a one-year period during the development of a large real-time multi-processor data processing system. In the graph on page 29 of reference [8] is a plot of errors detected per month. Below, Figure 40 gives the values (approximate) read off that plot and also lists the results of applying the extended Jelinski-Moranda model and Schneidewind's method 3. The parameters of these models were:

Sukert's Data [124]			
Week	Errors Found	Extended J-M	Schneidewind (Method 3)
1	67	143	140
2	106	140	135
3	97	136	130
4	129	132	125
5	77	127	120
6	142	124	115
7	116	118	111
8	131	114	107
9	187	108	103
10	203	101	99
11	136	92	95
12	183	87	92
13	45	80	88
14	46	78	85
15	71	76	81
16	54	73	78
17	57	71	75
18	80	68	72
19	64	65	70
20	27	63	67
21	42	62	64
22	55	60	62
23	62	58	59

Fig. 36 Extended J-M and Schneidewind (Method 3)

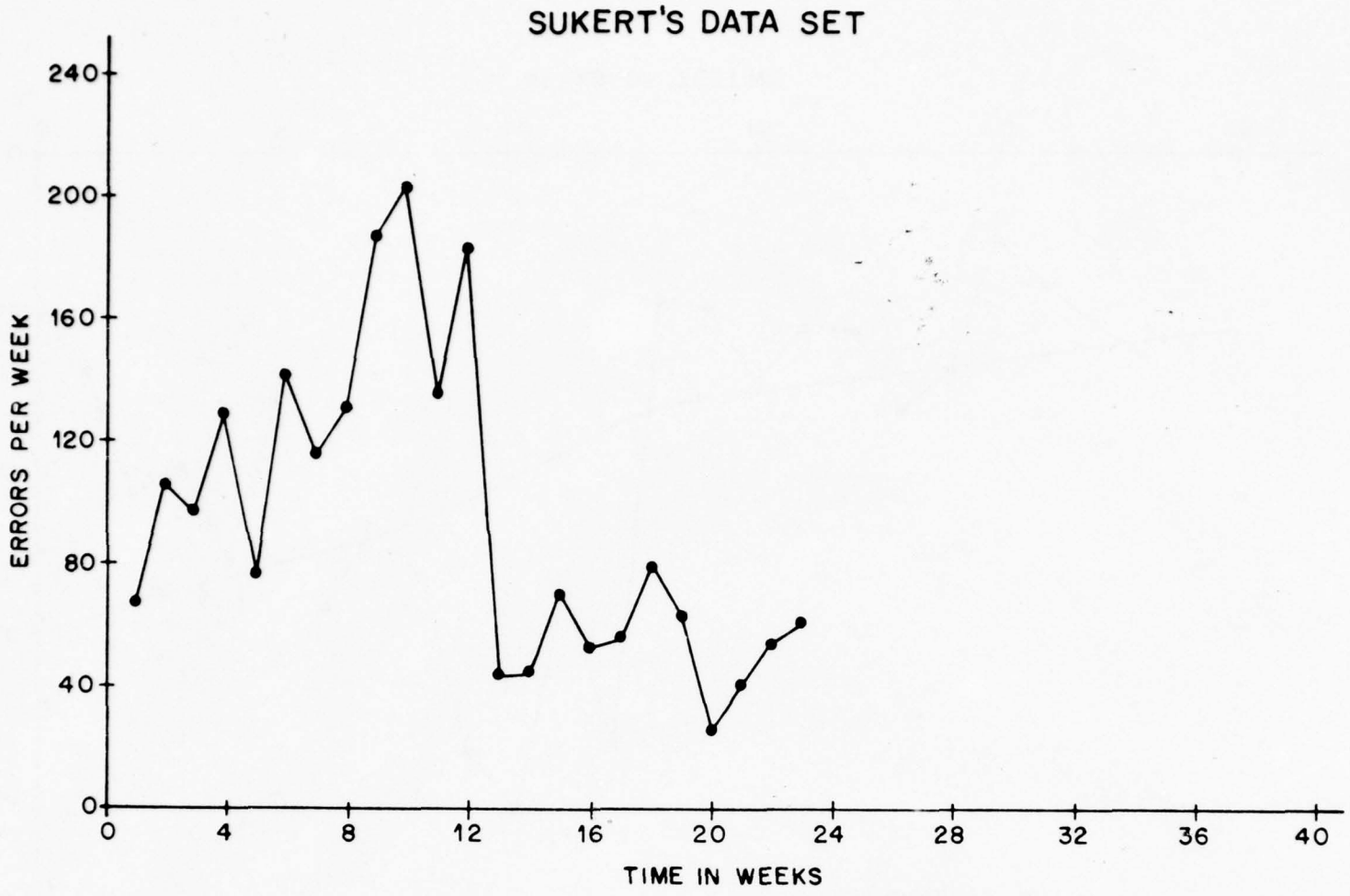

SUKERT'S DATA SET
ERRORS PER WEEK
240
200
160
120
80
40
0
0
4
8
12
16
20
24
28
32
36
40
TIME IN WEEKS

FIG. 37

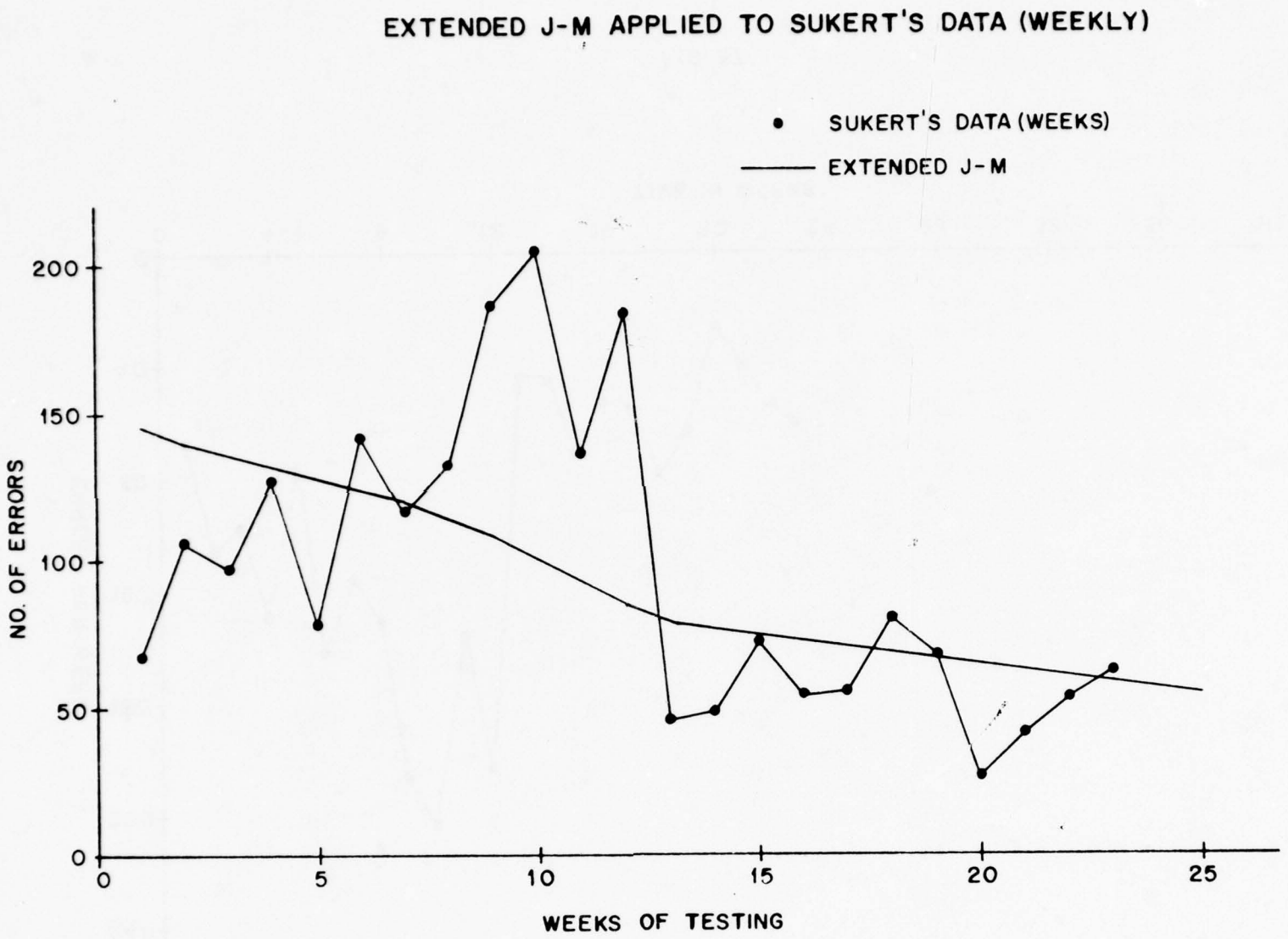

EXTENDED J-M APPLIED TO SUKERT'S DATA (WEEKLY)
SUKERT'S DATA (WEEKS)
EXTENDED J-M
NO. OF ERRORS
0
50
100
150
200
0
5
10
15
20
25
WEEKS OF TESTING

FIG. 38

	EXTJMEXP	EXTJMEXP	EXTJMEXP	EXTJMEXP	POISSON	POISSON	POISSON	POISSON
FROM:	1	6	10	12	1	6	10	12
TO:	23	23	23	23	23	23	23	23
67.0000	143.3773	0.0000	0.0000	0.0000	140.7272	0.0000	0.0000	0.0000
106.0000	140.6610	0.0000	0.0000	0.0000	135.3321	0.0000	0.0000	0.0000
97.0000	136.3635	0.0000	0.0000	0.0000	130.1438	0.0000	0.0000	0.0000
129.0000	132.4308	0.0000	0.0000	0.0000	125.1544	0.0000	0.0000	0.0000
77.0000	127.2008	0.0000	0.0000	0.0000	120.3562	0.0000	0.0000	0.0000
142.0000	124.0791	239.9123	0.0000	0.0000	115.7421	175.8054	0.0000	0.0000
116.0000	118.3220	230.2582	0.0000	0.0000	111.3048	161.6413	0.0000	0.0000
131.0000	113.6191	214.9845	0.0000	0.0000	107.0376	148.6183	0.0000	0.0000
187.0000	108.3080	201.0076	0.0000	0.0000	102.9341	136.6446	0.0000	0.0000
203.0000	100.7265	182.4190	205.8248	0.0000	98.9878	125.6355	151.9951	0.0000
136.0000	92.4964	171.3248	193.1864	0.0000	95.1929	115.5135	135.6247	0.0000
183.0000	86.9826	150.8638	173.1914	153.1521	91.5434	106.2069	121.0175	96.9311
45.0000	79.5633	134.1492	154.8940	139.1723	88.0338	97.6501	107.9836	89.6786
46.0000	77.7389	115.2732	130.5604	117.0550	84.6588	89.7827	96.3534	82.9687
71.0000	75.8739	88.3281	116.0357	96.8155	81.4132	82.5492	85.9759	76.7609
54.0000	72.9954	59.0775	89.2498	69.8991	78.2920	75.8984	76.7160	71.0175
57.0000	70.8061	39.4811	67.3684	53.8328	75.2905	69.7835	68.4534	65.7039
80.0000	68.4951	13.1123	42.6576	24.2039	72.4040	64.1613	61.0808	60.7879
64.0000	65.2517	6.6282	7.3032	.0000	69.6282	58.9920	54.5022	56.2396
27.0000	62.6570	.0000	-30.9092	-27.3337	66.9588	54.2392	48.6322	52.0317
42.0000	61.5624	-10.2305	-56.5632	-66.3520	64.3918	49.8693	43.3943	48.1386
55.0000	59.8596	-18.0114	-91.0830	-108.7088	61.9232	45.8515	38.7206	44.5368
62.0000	57.6297	-26.2246	-99.5715	-137.0857	59.5492	42.1573	34.5503	41.2045
	55.1161	-37.7520	-108.2486	-175.2694	57.2662	38.7609	30.8291	38.1216
	52.8815	-46.9738	-121.6415	-184.6589	55.0708	35.6380	27.5087	35.2692
	50.7376	-50.8643	-131.8277	-194.2570	52.9595	32.7668	24.5460	32.6304
	48.6805	-56.9161	-142.5798	-209.0714	50.9291	30.1268	21.9023	30.1889
	46.7069	-64.8412	-157.6704	-220.3387	48.9766	27.6996	19.5433	27.9301
	44.8133	-73.7748	-169.7429	-232.2320	47.0990	25.4679	17.4385	25.8404
	42.9964	-63.1445	-174.8359	-248.9243	45.2933	23.4161	15.5603	23.9070
	41.2533	-54.0459	-182.7585	-262.2782	43.5569	21.5295	13.8844	22.1182
	39.5807	-46.2584	-193.1333	-267.9119	41.8870	19.7949	12.3800	20.4633
	37.9760	-39.5929	-204.8285	-276.6753	40.2812	18.2001	11.0547	18.9322
	36.4364	-33.8879	-166.1912	-288.1513	38.7369	16.7338	9.8640	17.5157
	34.9592	-29.0050	-134.8421	-301.0073	37.2518	15.3956	8.8017	16.2051
	33.5419	-24.9256	-109.4065	-238.2646	35.8237	14.1460	7.8537	14.9926
	32.1820	-21.2484	-88.7688	-198.5498	34.4503	13.0063	7.0079	13.8709
	30.8772	-18.1067	-72.0241	-149.2081	33.1299	11.9584	6.2531	12.8330

Fig. 39 Extended J-M and Geometric-Poisson Models Applied to Sukert's Data

Baker's Data [8]			
Month	Errors Found	Extended J-M	Schneidewind (Method 3)
March	690	884	891
April	825	791	770
May	650	681	665
June	765	594	575
July	470	491	497
August	465	428	430
September	345	366	371
October	355	319	321
November	340	272	277
December	200	226	239
January	190	199	207
February	130	174	179

Fig. 40 Extended J-M and Schneidewind (Method 3)

extended Jelinski-Moranda exponential model:

$$N = 6592.86$$

$$\emptyset = 0.13405213$$

Schneidewind method 3:

$$\alpha = 957.629$$

$$\beta = 0.1458545$$

$$s = 4$$

Again, the results from the models were very similar where Figure 42 shows the extended Jelinski-Moranda model with the original data. It provides the best fit when the models performances are compared to the actual data using a chi-square test. An interesting aspect of this data is the long-range prediction both models give. The expected number of errors found per year can be tabulated as follows:

Year	Errors found during year
1 (given data)	5425
2	964
3	169
4	29
5	6

Fig. 41 Long Range Predictions

It is clear that a reduction in time between failures from 12 days, after 4 years of debugging, to 2 months, at the cost of a fifth year of testing would be hard to justify. Figure 43 plots percent deviation between the model predictions and the actual data for the Schneidewind method 3, extended Jelinski-Moranda and, additionally, the Geometric-Poisson. This reveals that the actual performance of the models is very close, at least for this data. This is indeed true in many cases. The Schneidewind method 3 gives a slightly better fit than the Geometric-Poisson or the equivalent Schneidewind method 1. The two model families give curves that are very similar. The extended Jelinski-Moranda model

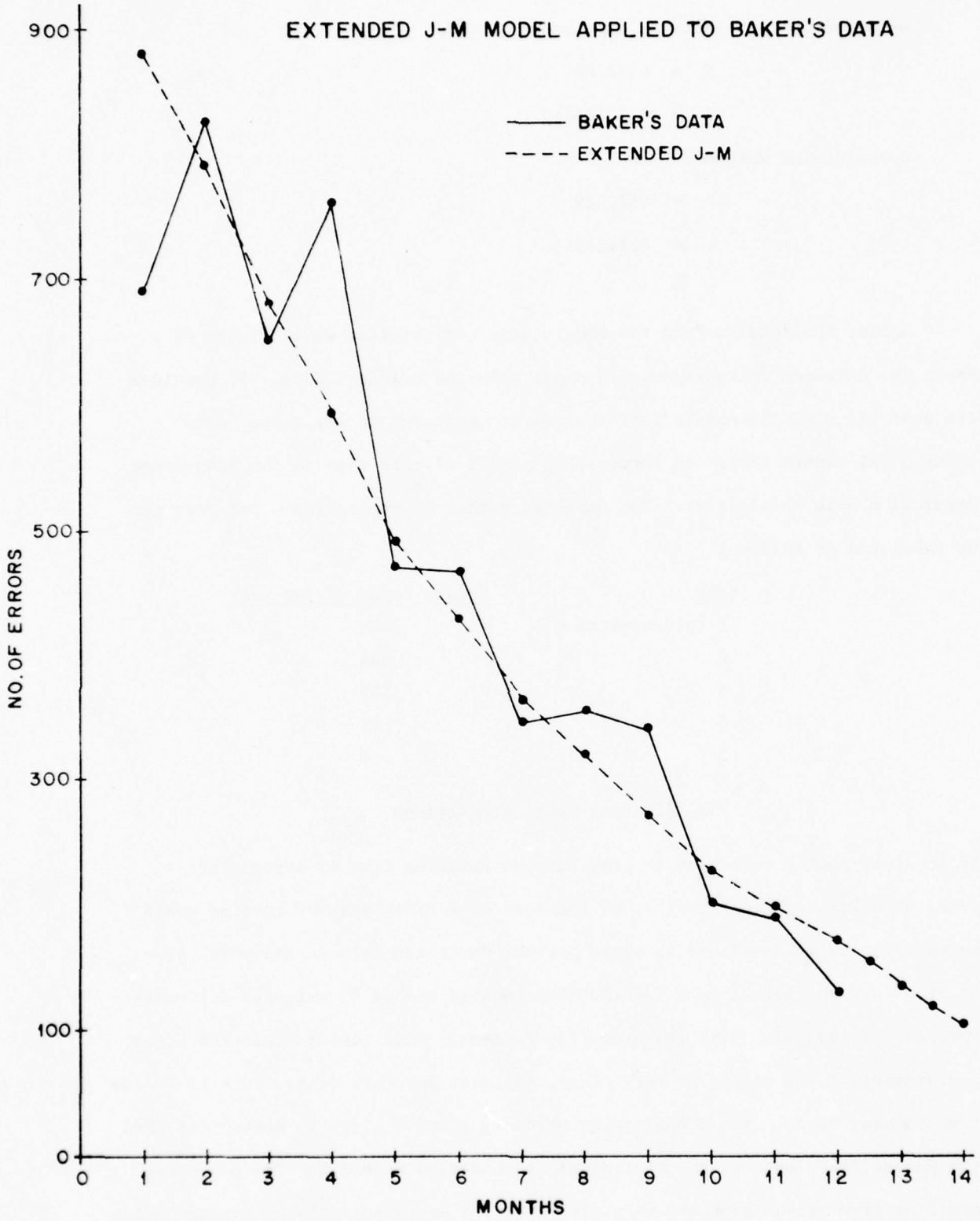

EXTENDED J-M MODEL APPLIED TO BAKER'S DATA
BAKER'S DATA
EXTENDED J-M
NO. OF ERRORS
900
700
500
300
100
0
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
MONTHS

FIG. 42

PERCENT DEVIATIONS IN MODEL PREDICTIONS FROM BAKER'S DATA

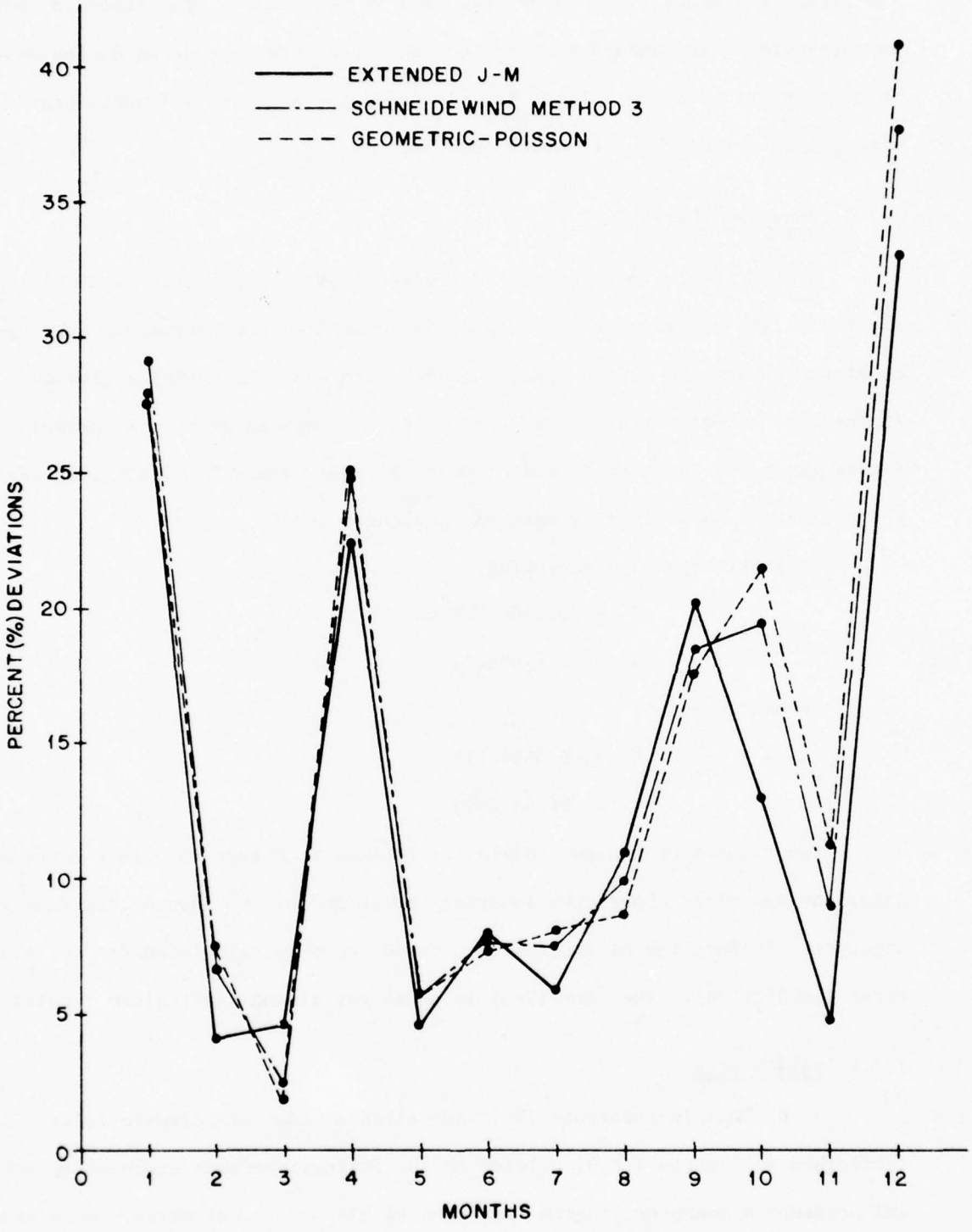


FIG. 43

gives an even better fit than the Schneidewind method 3, but has two disadvantages. The first is a sensitivity to erratic data as seen above. The second disadvantage, an outgrowth of the assumption that no new errors are introduced during debugging, is best described as optimism. In many cases the extended Jelinski-Moranda gives a much lower prediction of future bugs.

1.7.3 Honeywell Data

Damman et.al. in reference [27] describe the flight test of a multi-mode digital flight control system. Figure 44 summarizes the software errors encountered during the flight test, together with cumulative flying time and flying time between errors. Because the data is expressed as time between errors the basic Jelinski-Moranda exponential and Geometric models were applied. The parameters as derived by maximum likelihood were:

Jelinski-Moranda exponential:

$$\emptyset = 0.06083073$$

$$N = 4.37686774$$

Geometric model:

$$K = 0.54044111$$

$$D = 0.36422214$$

The results of the two models are plotted in Figure 45 with the actual data. As described above, the inherent "optimism" of the exponential family is apparent. In fact the hazard function cannot even be calculated for the sixth error (prediction). The term (N-n) is negative, giving meaningless results.

1.7.4 Musa's Data

J. D. Musa, in reference [83], describes a model of software reliability (described in Section 1.6.9) related to the Jelinski-Moranda exponential model and presents a computer program implementing his model. He gives, in an example,

Data from Honeywell Flight Test Program [27]			
Flight Number	T_i	X_i	Failure
2	4.0	4.0	Attitude hold step at change in algorithm at 45 - ratcheting (page 39)
3	6.2	2.2	LFP found unacceptably responsive for large inputs (page 122)
8	16.3	10.1	LPA software would not engage caused by prior change (page 124)
15	28.9	12.6	CPU 1 power supply failed: both CPU's went off line (page 98)
43	71.1	42.2	Deadspot caused by underflow, loss of significance in lateral trim integrator (page 109)

Fig. 44

J-M AND GEOMETRIC MODELS APPLIED TO HONEYWELL DATA

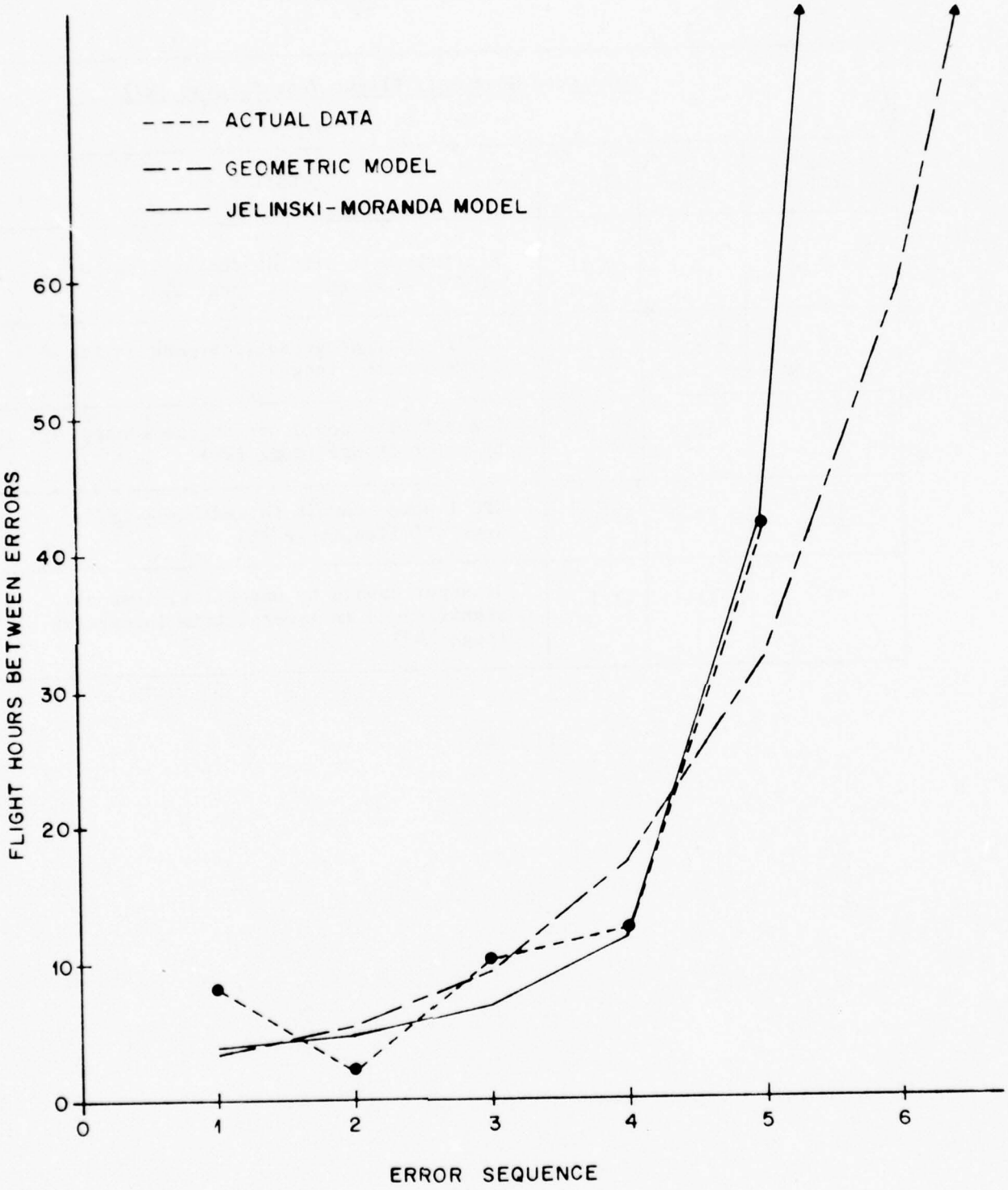


FIG. 45

central processor execution time between errors for 38 errors in a software project at Bell Laboratories. This data is plotted together with the Geometric model in Figure 46 and with the Jelinski-Moranda model in Figure 47. The model parameters were:

Jelinski-Moranda model:

$$\emptyset = 0.0000653488$$

$$N = 37.90032740$$

Geometric model:

$$K = 0.88858641$$

$$D = 0.01042702$$

As with the Honeywell data, the Geometric model gives a reasonable fit. It is also fairly consistent in its results when only part of the data is used. The Jelinski-Moranda model, in contrast, runs into trouble with its "no new errors" assumption. As can be seen in Figure 47, it shows a sudden sharp rise in the time between errors, and indicates by the N value that the 38^{th} error was the last one in the software. This does not seem to fit the data, and in our experience with this model it has always been more optimistic than the Geometric. The larger the data set, the more likely it is to claim a sudden improvement and suggest that debugging is complete. This is illustrated by the table in Figure 48 giving model solutions using various portions of this data set. At this time, the Geometric model appears to be slightly better than the Jelinski-Moranda model for data in terms of time between errors.

1.8 A METHOD OF INCREASING SURITY IN MODEL APPLICATION

In Section 1.2 we saw the undesirability of total program testing (i.e. checking all possible paths through a program), and the prohibitive complications involved in formal, inductive proofs of correctness for large programs. The seeding and tagging techniques, discussed in Section 1.3, though they show some promise, are not at an advanced enough stage of development to be used in our analysis.

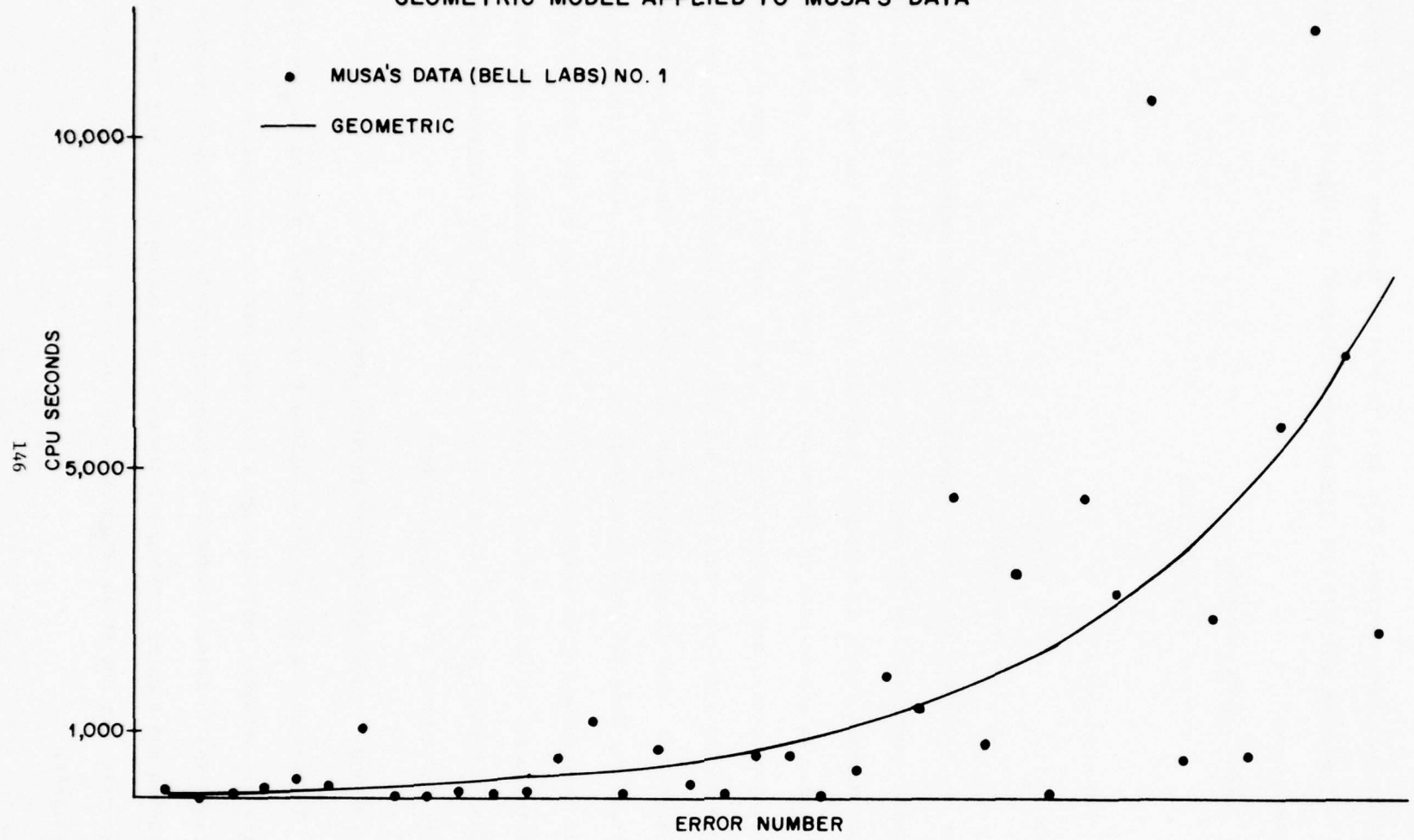

GEOMETRIC MODEL APPLIED TO MUSA'S DATA
MUSA'S DATA (BELL LABS) NO. 1
GEOMETRIC
10,000
5,000
1,000
CPU SECONDS
ERROR NUMBER

FIG. 46

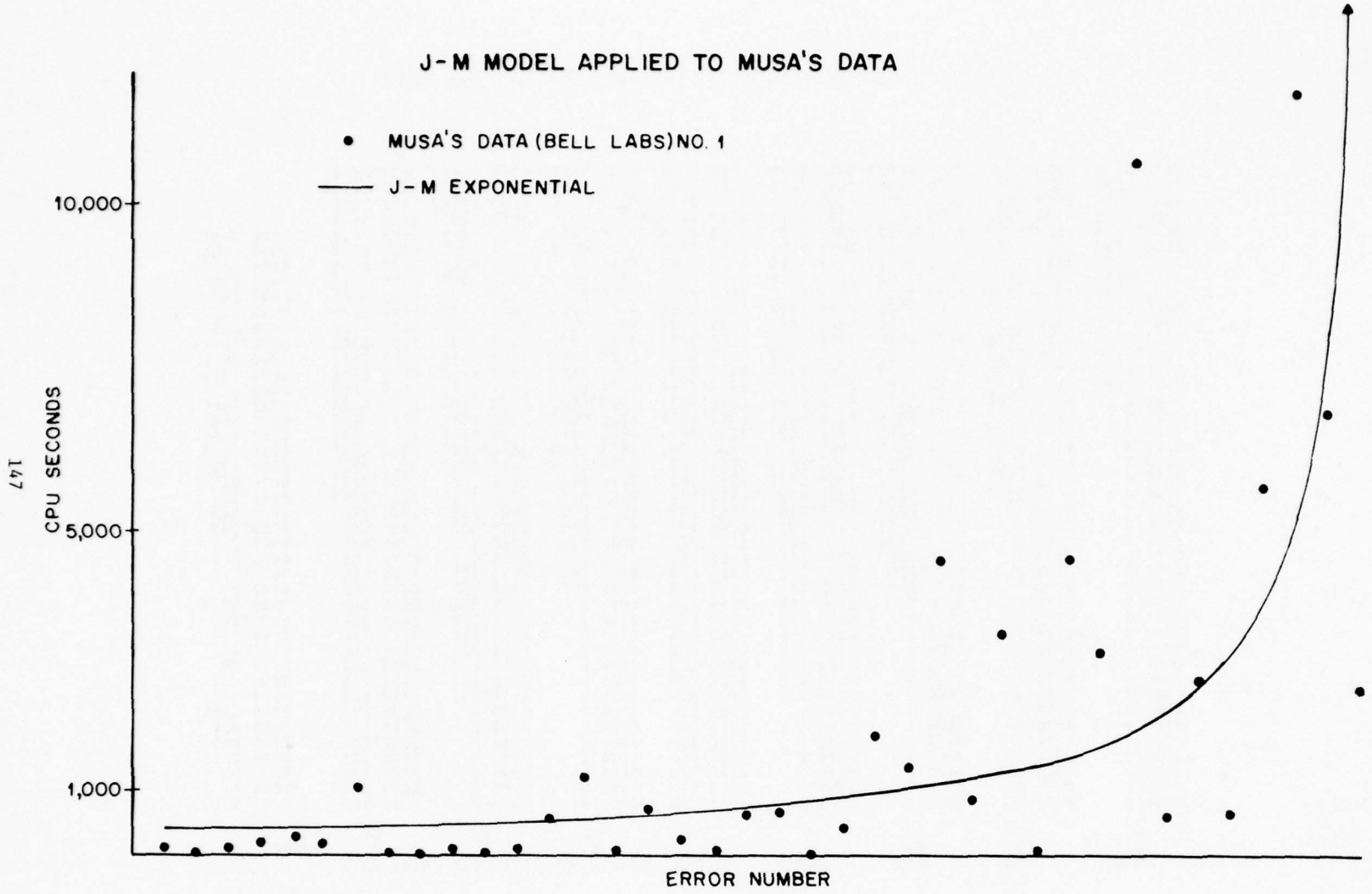

J-M MODEL APPLIED TO MUSA'S DATA
MUSA'S DATA (BELL LABS) NO. 1
J-M EXPONENTIAL
10,000
5,000
1,000
CPU SECONDS
ERROR NUMBER

FIG. 47

PRINT MTTE

FROM: / TO:	GEOMETRIC 1 38	GEOMETRIC 8 38	GEOMETRIC 15 38	JMEXP 1 38	JMEXP 8 38	JMEXP 15 38	JMEXP 21 38	JMEXP 28 38
115.0000	95.9847	0.0000	0.0000	403.7560	0.0000	0.0000	0.0000	0.0000
0.0000	107.9295	0.0000	0.0000	414.6978	0.0000	0.0000	0.0000	0.0000
83.0000	121.4628	0.0000	0.0000	426.2491	0.0000	0.0000	0.0000	0.0000
178.0000	136.6912	0.0000	0.0000	438.4623	0.0000	0.0000	0.0000	0.0000
194.0000	153.8340	0.0000	0.0000	451.3961	0.0000	0.0000	0.0000	0.0000
136.0000	173.1176	0.0000	0.0000	465.1161	0.0000	0.0000	0.0000	0.0000
1077.0000	194.8236	0.0000	0.0000	479.6963	0.0000	0.0000	0.0000	0.0000
15.0000	219.2512	114.3142	0.0000	495.2202	560.2979	0.0000	0.0000	0.0000
15.0000	246.7416	132.9035	0.0000	611.7825	578.8541	0.0000	0.0000	0.0000
92.0000	277.6787	154.6158	0.0000	629.4989	598.6815	0.0000	0.0000	0.0000
50.0000	312.4949	179.6425	0.0000	548.4687	619.9164	0.0000	0.0000	0.0000
71.0000	351.6765	208.8552	0.0000	568.8574	642.7109	0.0000	0.0000	0.0000
606.0000	395.7707	242.8184	0.0000	590.8206	667.2468	0.0000	0.0000	0.0000
1189.0000	445.3936	282.3045	0.0000	614.5478	693.7305	0.0000	0.0000	0.0000
40.0000	501.2383	328.2117	306.8596	640.2606	722.4034	898.6259	0.0000	0.0000
788.0000	564.0851	381.5842	357.3018	668.2189	753.5486	936.1075	0.0000	0.0000
222.0000	634.8117	443.6368	416.0357	698.7304	787.5004	976.8520	0.0000	0.0000
72.0000	714.4063	515.7781	484.4245	732.1617	824.6560	1021.3048	0.0000	0.0000
615.0000	803.9807	599.6518	564.0552	768.9527	865.4914	1069.9961	0.0000	0.0000
589.0000	904.7861	697.1647	656.7757	809.6369	910.5816	1123.5627	0.0000	0.0000
15.0000	1018.2309	810.5348	764.7378	854.8667	960.6283	1182.7753	1632.6595	0.0000
300.0000	1145.8997	942.3407	890.4469	905.4489	1016.4961	1248.5762	1713.5566	0.0000
1863.0000	1289.5759	1095.5803	1036.8204	962.3935	1079.2636	1322.1297	1802.8884	0.0000
1337.0000	1451.2668	1273.7391	1207.2558	1026.9814	1150.2927	1404.8918	1902.0466	0.0000
4508.0000	1633.2309	1480.8693	1405.7060	1100.8621	1231.3297	1498.7072	2012.7471	0.0000
834.0000	1838.0102	1721.6822	1636.7787	1186.1967	1324.6500	1605.9487	2137.1294	0.0000
3400.0000	2068.4653	2001.6551	1905.8357	1285.8727	1433.2755	1729.7206	2277.8974	0.0000
6.0000	2327.8156	2327.1561	2219.1208	1403.8369	1561.3076	1874.1640	2438.5171	3081.3713
4561.0000	2619.6840	2705.5887	2583.9043	1545.6310	1714.4572	2044.9296	2623.5068	3254.5703
3186.0000	2948.1477	3145.5606	3008.6517	1719.2870	1900.9196	2249.9337	2838.8671	3448.3994
10571.0000	3317.7951	3657.0787	3503.2198	1936.9037	2132.8902	2500.6208	3092.7470	3666.7779
563.0000	3733.7980	4251.7778	4079.0861	2217.5928	2429.3450	2814.1756	3396.4956	3914.6851
2770.0000	4201.9436	4943.1848	4749.6145	2593.4221	2821.5129	3217.6376	3766.4065	4198.5445
652.0000	4728.7957	5747.0251	5530.3656	3122.6346	3364.6697	3756.1482	4226.7388	4526.7884
5593.0000	5321.7060	6681.5828	6439.4581	3923.2012	4166.8822	4511.1425	4815.2630	4918.7096
11696.0000	5988.9572	7768.1145	7497.9892	5275.7817	5471.1061	5646.0017	5594.1886	5365.7872
6724.0000	6739.8703	9031.3335	8730.5236	8051.7334	7964.0282	7543.7738	6873.7458	5913.8237
2646.0000	7584.9350	10499.9721	10185.8645	16992.8307	14630.4868	11363.2739	8269.5992	6586.5426
	8535.9565	12207.4346	11836.7168	**********	89790.7811	23017.0973	10868.5255	7431.9542
	9606.2199	14192.5577	13782.4600	-13918.1408	-21702.8971	**********	15849.6798	8526.3485
	10810.6761	16500.4934	16048.0483	-7288.7791	-9681.4235	-21897.3057	29259.6892	9998.7112
	12166.1505	19183.7362	18686.0586	-4937.1574	-6230.3690	-11083.4566	90089.8965	12085.7198

Fig. 48 J-M and Geometric Models Applied to Musa's Data

This led to our accepting a stochastic process for software error occurrence and to the conclusion that certain analytical models (vis-a-vis, structural and empirical models in Sections 1.4 and 1.5 respectively) can best represent the software reliability. Debugging, the iterative process of exercising the software until an error is discovered and removed, is the most widely used method of increasing software reliability. Debugging continues until the occurrence of errors becomes so infrequent that the software is considered "acceptable". The analytical models presented earlier attempt to measure and predict the reliability. However, what would be desirable would be a method which determines (1) how much confidence can be placed in the model results and (2) when the model results are accurate enough for the debugging process to be stopped. Such a method would indeed be useful in the study of software reliability.

A very recent development which addresses these very topics was presented by Forman and Singpurwalla [37]. They point out that although the method of maximum likelihood gives better results than either the least squares or the Bayesian methods in estimating model parameters, there can be difficulties with an indiscriminant application of it. They also present an empirical stopping rule for debugging the software.

The Jelinski-Moranda model has the hazard function $\emptyset[N-(i-1)]$ where N is the total number of initial errors and $\emptyset$ is the constant of proportionality. The data required for this model is the sequence of times between errors (i.e. $t_1, t_2, t_3, \ldots\ldots, t_n$). N and $\emptyset$ are the model parameters and the best method of estimation is maximum likelihood. However, if n (the total number of errors discovered to date) is very much less than N, then it is possible for the ratio of the weighted sum of time between errors ($\sum_{i=1}^{n} (i-1)\, t_i$) to the total debugging time ($\sum_{i=1}^{n} t_i$) to be very small. This in turn can cause unstable and misleading values of N and $\emptyset$, the maximum likelihood estimators of $\hat{N}$ and $\hat{\emptyset}$ respectively.

To guard against such a situation Forman and Singpurwalla suggest comparing the graphs of the following two functions:

(1) R(N), the relative likelihood function of N

(2) $R_{normal}(N)$, the normal relative likelihood function for N.

Large-sample theory ensures that N is approximately normally distributed with mean N. R(N) measures the relative plausibility of various values of N and its most plausible value $\hat{N}$. $R_{normal}(N)$ represents the distribution of the relative likelihood under the large-sample condition. A comparison of R(N) and $R_{normal}(N)$ will give some indication of how accurate an estimate of N is. It will also be possible to determine confidence limits for N and $\emptyset$.

In light of the above results, the authors present the following stopping rule for the debugging of software:

(1) Compute $\hat{N}$, the maximum likelihood estimator of N.

(2) If $\hat{N} = n$ (or very close to it), proceed to Step 3; if $\hat{N} >> n$, observe another failure, record the time (t_{n+1}) for that failure to occur, and go to Step 1.

(3) Compute R(N) and compare it with $R_{normal}(N)$. If a plot of the two functions shows a large disparity, the observed estimator is misleading. Observe another failure time (t_{n+1}) and go to Step 1. If a plot of the two functions is in good agreement, stop testing and accept the software.

The findings of Forman and Singpurwalla are quite pertinent to any study of analytical software reliability models which utilize maximum likelihood techniques for parameter estimation. They clearly show that care must be taken in the parameter estimation, and that great confidence cannot be given to the estimators unless certain conditions are first met. Their ideas are a definite contribution to the development of a means of assuring greater reliance on the software reliability models.

The stopping rule for debugging seems to have promise in aiding software developers in their decision of "how good" their software is at any given stage of testing. However, we feel that before it can be made extensively applicable,

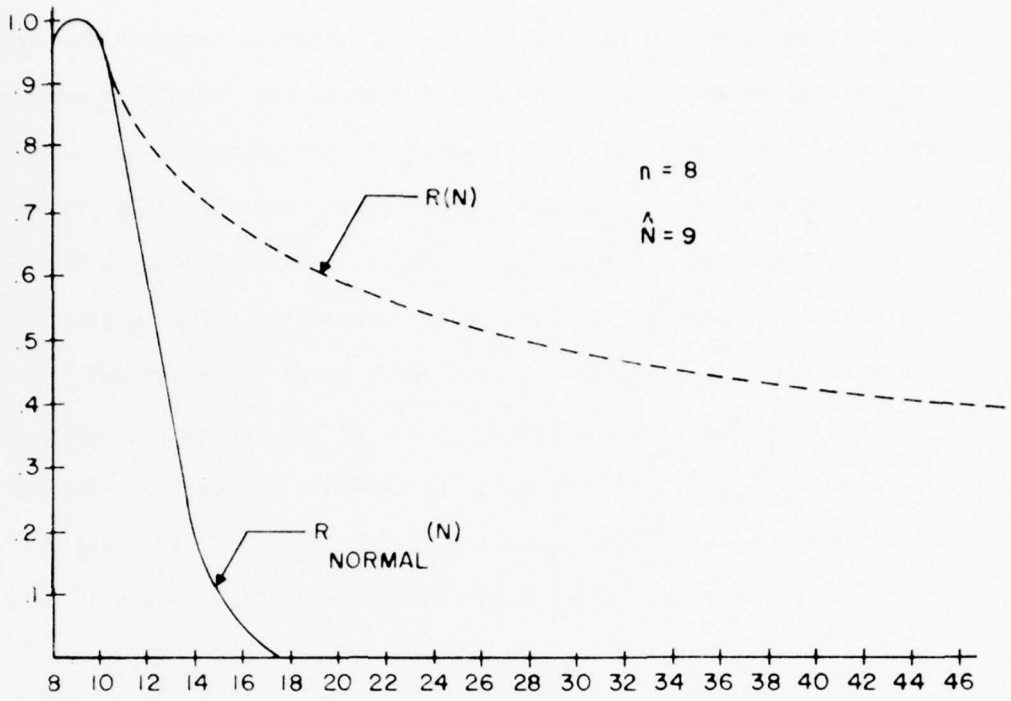
it will be necessary to quantify the relationship between R(N) and $R_{normal}(N)$. Presently, it is a subjective decision as to how "good" the agreement between the two curves is. The comparison of the two curves is critical in the stopping rule, and what may appear good to one person may not be so for someone else.

Forman and Singpurwalla did apply their technique to the F-11D data program given by Wagoner in [134] and reproduced as Figure 6 (Section 1.6.2) in this report. If testing were terminated after the first day (1/12), Figure 49 would be derived with $\hat{N} = 9$ and $n = 8$. As can be seen, a large disparity between $R_{normal}(N)$ and R(N) is quite evident. Hence, this implies that debugging should be continued. Finally, if the procedure is applied after the last interval (1/31), we find that $\hat{N} = 107$ and $n = 107$. Figure 50 does show a very good agreement between the two derived curves.

Of course, the stopping rule is designed for a test situation, but very often we are presented with a situation in which we are simply handed a set of test data after the testing has ended, with no chance to test further. If upon applying one iteration of the stopping rule we find that the parameters estimators are inadequate and that further testing is needed, we are stuck. We do in fact know that such model results are poor, but we can do nothing to improve them in these cases.

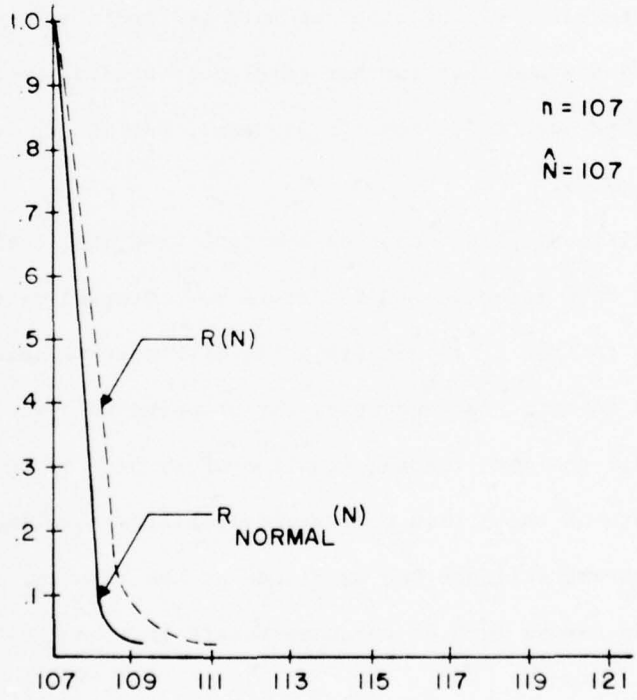
One additional point deserves mention. The most common form in which software error data is collected is errors per interval rather than time between errors. In fact it is usually quite difficult to obtain data in the time between errors format. Consequently, the stopping rule would have to be adjusted to handle the more common situation of errors per interval.

The state-of-the-art in software reliability modeling is not yet at a stage where extreme reliance can be placed on the results. Yet Forman and Singpurwalla do remove much of the uncertainty that once existed.



RELATIVE LIKELIHOOD AND LARGE-SAMPLE NORMAL APPROXIMATION USING F11-D DATA (1/12)

FIG. 49



RELATIVE LIKELIHOOD AND LARGE-SAMPLE NORMAL APPROXIMATION USING F11-D DATA(1/31)

FIG. 50

SECTION II

SOFTWARE ERROR DATA

2.1 INTRODUCTION

In order to construct and apply any of the models that have been presented, it is imperative that a "clean" and meaningful data set be available. To date, the availability of usable software error data has been a major problem which has plagued many of the software model developers. Jelinski and Moranda describe this problem:

"It was surprising to find that there was no such thing as software failure data collection for analysis' sake. The obvious place to look was inside the company where many software systems are developed; strangely enough, nobody is interested in software failures, though everyone in concerned with software reliability. It may be a paradox, but most 'bugs' are fixed without failures being documented, and yet perfect software is required."

When Jelinski and Moranda did find data, they found that it was not available in a usable form, being either classified, summarized, or vague. Compounding this problem is the reluctance of manufacturers to release data, reflecting perhaps their unwillingness to admit that they, as do all software designers, make mistakes.

In the forthcoming discussion, these problems, as well as other data acquisition problems, will be analyzed.

2.2 ERROR DATA: NECESSITY AND PROBLEMS

Initially, it should be established that acquiring quality software error data in sufficient quantities has been a formidable task. Not only will such data be useful in modeling studies, but it may also be beneficial in planning future software projects. For example, error data might be utilized in predicting the amount of testing time required, assessing the costs associated with testing, and estimating types of errors by category. However, it appears

as though the availability of well documented error data would be most advantageous in evaluating the viability of the numerous software reliability models proposed to date. Without such data, accurate appraisals of many of these models cannot be made, and subjective judgment is forced to play a major role. Jelinski and Moranda [56] assess the overall problem and comment "the importance of software reliability in our strategic defense systems and in our space systems cannot be overemphasized". Finally, they hypothesize that given a good set of data, it would be possible to develop a fairly accurate model, which would include the computation of confidence limits and the capability of predicting future software failures.

Hence, the availability of error data would not only serve as a basis for evaluating the models proposed to date, but it would also support the development of newer models. Therefore, it must be reiterated that the need for well documented software error data cannot be emphasized strongly enough. In fact, the Rome Air Development Center (RADC) is attempting to build up a software data repository, which hopefully will be of considerable use in software reliability studies. In early 1976, RADC awarded contracts to various organizations, including IBM/Federal Systems Division, Boeing Aerospace Company, Raytheon Company, and The Charles Stark Draper Laboratory, Inc. The primary purpose of the aforementioned contracts was to acquire software error data from large software projects. This research is mandatory if reliable, maintainable, and quality software is to be produced in both the industrial and military environments. Baker [8], Fries [40], Willman et. al. [137], and Rye et. al. [105] discuss their respective approaches, problems, and conclusions regarding the acquisition of adequate software error data.

Numerous problems, in both these recent efforts and in prior studies, are existent and require further investigation. Often, during the data collection

process of recent software projects, the following comments were commonplace. Thayer et.al. [128] believe "presently data collection schemes don't provide enough of the right kind of data, nor are the data collected at the right time". Baker [8] states "some of the data collected during the SDA study was not as valid as had been hoped. The major portion was collected after the fact, and much of the source information needed had already been disposed of or had never been present. Other factors that would have contributed to the study became apparent during the course of the contract". Hence, it becomes readily apparent that when error data has been collected it oftentimes was in too general a form or not the right type of data. Improper planning is directly responsible for these and many other problems encountered in data acquisition studies. Therefore, in the future, poor planning must be avoided at all costs since the majority of the time data cannot be reconstructed later on if it was not collected initially.

In the literature, many authors have strongly agreed that software projects do have the potential to produce large amounts of quality software error data. However, as was stated, some of the studies conducted in the past, in which a great deal of effort was expended, produced results which were not as useful as had been anticipated.

On the other hand, some investigations have realized moderate success. Shooman and Bolsky [114] mention "it was shown that such a study can be done, that meaningful data can be obtained and analyzed". Fries [40] is of the opinion that not only can useful data be collected but it must be collected.

Furthermore, complicating the modeling and data collection process, is the problem of selecting an appropriate time base. That is, data may be available only with respect to calendar time, which is not at all likely to be an appropriate time frame for software. In fact, later in this section, it will be seen that many of the data sets available in the literature are presented in terms of calendar time. In many of these cases, model applications

to such data produced poor results (predictions), and, in our opinion, the time scale was one of the contributing factors.

Hence, a time base which reflects either the "age" of the software in terms of execution time, or the amount of debugging effort (manhours) that have been applied to it is clearly more appropriate. Furthermore, there is no reason to expect that calendar time would have any clear relationship to these alternate measures. As a result, a careful analysis of the data would be required to either scale or weight the data to an appropriate scale as preparation for applying a particular software model. We do not mean to suggest that there is an easy solution to this problem; on the contrary, extreme caution should be exercised and thorough consideration should be given to the available data. The success or failure of a model may depend on the selection of an appropriate time reference. It is our opinion that CPU or operational time is the most appropriate time base to employ, especially when evaluating the reliability of an operational piece of software.

How can these aforementioned problems be alleviated? One initial question to be considered is: How should the data be collected and at what level of detail? The answer to this question has not been clearly established at this point in time to satisfy all researchers. Thayer et.al.[128] propose the following approach.

(1) Determine what data is available.

(2) Consider what methods can be utilized in collecting and storing these data.

(3) Establish criteria for documenting software errors.

(4) Consider approaches for characterizing the software, and the development and test processes in quantitative terms.

(5) Specify methods of analysis.

Fries [40] suggests the following, which is more specific than that presented above:

(1) Define categories for various types of errors.

(2) Identify the source of the error.

(3) Specify the type of correction made.

(4) Record the time to both find and fix the error.

Although both approaches are appealing in some respects, a well defined and universally accepted procedure would be desirable. However, it would be highly probable that problems would continue to arise. As previously discussed, software error data may be used by different groups, each of which may have a unique goal or need for such data. Thus, while a method of acquisition derived for modeling analysis may be attractive to researchers in that area, at the same time it may not satisfy certain requirements for groups whose interests lie in areas other than modeling.

Endres [34] believes "it may be beneficial to design a set of questions pertaining to the errors so that you may identify what you need to collect". It is our belief that exactly what needs to be collected must be clearly specified prior to undertaking an intensive data collection program. First, a particular procedure or format for data collection should be established. Then, comparing the proposed approach with the requirements of various software models should indicate whether everything necessary will, in fact, be collected so that meaningful model applications are possible.

Specifically, there are a multitude of factors to seriously consider when identifying requirements for such studies. Perhaps the aspect which impacts the data collection process the most deals with programming personnel. The amount of time and effort expended to accurately fill out error reports or other related forms must be minimized. Their time is, and must be, primarily applied toward software production, testing, and verification. Hence, due to

such demanding schedules and the costs involved, it can be extremely difficult to accurately collect data in the development process. Therefore, the acquisition of software error data must be a process that takes little time yet is also effective in achieving requirements. It appears as though a checklist type form is more acceptable than a descriptive writeup of errors. One immediate advantage is that it would be less time consuming if it were well designed. Among others, Shooman and Bolsky [114] and Wagoner [134] advocate the use of the aforementioned type format. Wagoner presents what is called a Job Analysis Sheet which was designed to be one page in length so as to minimize the inconvenience to the programmer reporting the error. This point is supported by the Raytheon study where a common complaint among programmers was that the forms being used were too detailed. It should be pointed out however, that in Shooman and Bolsky's study, the programmers believed that the data collected was both valid and useful, and that the amount of time to fill out the error reports was not excessive. Thus, the importance of the design of the error reporting form to be completed is very evident as the comments from the Raytheon and Shooman-Bolsky studies indicate. A strong attempt must also be made to record and document the errors as completely as possible. Two points to be considered here are (1) prior to commencement of the project, check for proper understanding of the error report forms, and (2) explain to the involved personnel why the data is being collected. Good documentation is necessary, as in one particular study many errors were recorded so generally that they had to be eliminated from the study, thus distorting the "true" error process.

This leads to the problem of how should error reporting forms be designed and what should be included. One approach presented [128] was:

(1) When was the error introduced?

(2) How critical is the error?

(3) How and when was the error found?

(4) How much testing was done?

(5) Was the error independent of other errors or the result of a previous fix?

In our experience with a particular manufacturer, the following format for collecting error data was deemed mutually acceptable:

(1) Identify the program function where the error was discovered.

(2) Classify the error by type (coding, specification, etc.).

(3) Determine and categorize the criticality of the error.

(4) Estimate the operating time since the last error in that segment/function was discovered.

The fourth point listed above brings up an interesting question. That is, how useful are these estimates going to be in a modeling study? It may be recalled that in the discussion of the individual models, some of them required time-between-failures as an input (Geometric, Jelinski-Moranda, etc.). Then, in order to be able to apply this type of model, data of this type needs to be obtained. To date, very little data available in the literature was recorded in this manner. However, this would be the ideal way to record errors in our opinion. Below are our reasons for believing this:

(1) Data in the form of time-between-failures would be applicable to the Geometric, Schick-Wolverton, etc. models.

(2) This data could be converted to errors per interval to input the extended Jelinski-Moranda, Geometric-Poisson, etc. models.

(3) Model evaluations could be made on the basis of application to a single data set.

Data recorded in the form of errors per interval is useful only to the models mentioned in point two above. That is, such data cannot be converted back to time-between-errors. If one wished to determine which models were the best predictors, assessment of their capability could most easily be

evaluated by application to a single set of error data. Thus, although time-between-errors appears to be the better measure, most recorded error data sets are not of this type.

It has also been observed that most organizations participating in software error collection studies firmly believe that error categorization is a necessary requirement. However, extreme caution must be exercised so that the categories are defined such that no discrepancies occur as to what category an error should be assigned to. In other words, if too few categories are defined, errors may occur which do not correspond to an appropriate category. On the other hand, defining an excessive number of categories may present problems in that an error might "seem" to fall into more than one category. A TRW study showed that reducing the number of categories originally specified proved more usable in the data collection project and also was much more convenient to the programmers collecting the data. Thus, it would seem as if some basic categories could be established for all software projects with additional categories set up for individual projects if necessary. Now, some of the benefits of the categorization process may be realized. First, it may be possible to identify those areas which cause the most problems in terms of costs and effort expended. Second, the percentage of errors in various categories (e.g. logic - 15%, I/O - 2%, etc.) may be helpful in estimating the number of errors in these respective categories in future projects. One final suggestion concerning error categorization is that since some judgment may be involved, it might be beneficial to obtain concurrence with a supervisor or other personnel regarding the correct classification of certain errors.

Finally, it is mandatory that only software errors be included in the collection and categorization process. Records on hardware errors, user requested changes, and other "non-software" errors may be kept if desirable.

Baker [8] sums up this particular area very well by saying "the proper assignment of error categories is a key to the study of error reliability modelling".

Another problem to be aware of is the recording of duplicate error reports. It is our belief that each error should only be counted once, and this is in agreement with the assumption made by many of the software reliability models. A study conducted by one organization indicated that duplicate problem reports occurred frequently in a few select categories. In such instances, it was necessary to insure that duplicate reports were ignored so the error was counted only once. The problem of error generation during the debugging process has been and continues to be a problem also. While many of the models make the assumption that no new errors are added during debugging, in reality, there is a distinct probability that fixing other errors may introduce additional software failures.

The Raytheon data acquisition study [128] identifies several other characteristics of a software system which may cause difficulties with the interpretation of error data from that system. Our experience reinforces the fact that careful consideration must be given to these items in the analysis of software data.

(1) Evolutionary development of software requirements and of the system itself -- Large software systems do not just hatch overnight, but evolve in a step by step fashion over a long period of time. During planning, the requirements may change often, and during development, the system will change as well. Thus, it may be difficult to determine the moment when the developing system actually "becomes" the system and when the error history becomes applicable.

(2) Parallel hardware development -- Oftentimes the hardware undergoes changes during software development which may cause a software error that would not otherwise be experienced.

(3) Multiple system configurations -- The software, or portions of the software, may be tested and verified at different facilities which have slightly different functions (e.g. I/O). Thus, an error experienced at one facility may be unique to that facility and not to the software itself.

(4) Build process -- The reliability of a piece of software will often fluctuate with successive builds or integrations to the system.

(5) Uneven application of resources -- The number of errors discovered will be dependent on the number of debuggers working or on "how hard" they are working.

(6) Previously existing software -- In some software systems, certain portions may not be entirely new but merely modified from previously written packages which may already be debugged.

Endres [34] summarizes the problem of software error data acquisition well. He states "analysis of errors is a difficult task, yet it is a necessary and useful activity". This does seem to be very true, although it is hoped that future data collection studies will be well designed and carefully planned so that meaningful data will be available for model evaluation and future model development.

2.3 DATA SETS EXTRACTED FROM THE LITERATURE

Below is an individual listing of the software error data sets found in various papers and reports during the course of our study. The author(s), reference number, and type of data are given, as well as some brief comments on the utility of the respective data sets. Our comments are not intended to

be critical or reflect negatively on these data collection studies, but are merely our interpretation and evaluation of the usefulness of each data set as related to our objectives.

(1) W. T. Baker [8]; 5446 errors are distributed among 12 monthly intervals. A graph of errors per month is given although no numerical tables are presented. Because the graph was large, it was possible to read off it to obtain fairly accurate estimates of the number of errors per month, and the more promising models were then applied to this data.

(2) J. de S. Coutinho [22]; errors are listed in 36 weekly intervals. Erratic behavior is quite evident and may in part be due to the fact that calendar time was used as the time scale. Thus, there is no way to determine how much time was actually expended per week.

(3) L. Damman et.al. [27]; five errors were extracted from this report in the form of time-between-failures. Although the data set was extremely small, it was one of the few sets of data where time-between-failures was available. Hence, this data set was used with those models requiring such data.

(4) M. J. Fries [40]; 2036 errors are listed by category. The data appears to be in a summarized form although it is felt that if the raw data were available model application would be possible.

(5) M. Lipow and T. A. Thayer [65]; 25 groups of software problems are presented in this paper. It was felt that this data set was in a form not suitable for application and was not pursued since "better" data sets were available.

(6) B. Littlewood and J. L. Verrall [67]; 80 times-between-failures are printed out in a table. However, this data was generated through simulation and thus could not be classified as "real" error data.

(7) I. Miyamoto [77]; graphs are presented in this paper, however no numerical data is given to supplement these graphs. Due to this limitation it was felt that an evaluation of the models was not possible with this particular data set.

(8) P. B. Moranda [79]; two sets of data are available. The first set contains 3272 errors broken down in six monthly intervals. Set number two recorded 1905 errors distributed among five monthly intervals. These sets were somewhat restricted since the few monthly intervals given could not be further broken down and also since calendar time was employed as the time scale.

(9) J. D. Musa and P. A. Hamilton [83]; two sets of data were available in this report also. Both sets were recorded in terms of time-between-failures. The first set contained 38 points, was useful in our model application, and reasonably good results were derived. Set number two, comprised of 53 data points, was much more erratic and poorer results were obtained when the models were applied.

(10) P. Rye et.al. [105]; 11,729 modifications are given and were obtained from the Apollo program. It was our interpretation that these modifications were not all software errors but included enhancements or modifications to the program, and it was not possible to separate out the software errors from the enhancements.

(11) N. F. Schneidewind [108]; this paper presents graphical data but no numerical results are included. It was also difficult to read accurately off the graph due to the scale of each axis.

(12) N. F. Schneidewind [109]; only five data points are presented, and it was our opinion that this data set was too small for a meaningful application to be made.

(13) M. L. Shooman [114]; seven sets of data are presented, and although an adequate number of errors are available with each data set, the intervals are in months and cannot be broken down further. Further, calendar time was used as the time scale, which somewhat detracted from the attractiveness of the data.

(14) A. N. Sukert [124]; 2191 errors were grouped by days. To reduce the number of intervals, weekly intervals were established for model application. Calendar time was employed as the time frame and erratic behavior was very noticeable.

(15) W. L. Wagoner [134]; 107 errors were observed and could be grouped into errors per interval. This was one of the better data sets available since CPU time was used. Thus, some of the software models were applied to this data.

(16) H. E. Willman et.al. [137]; 2165 Software Problem Reports (SPR's) were listed by category. This report did give a graph of SPR's per month but numerical data was not included in the summarized listing.

SECTION III

CONCLUSIONS AND RECOMMENDATIONS

3.1 MODEL CONCLUSIONS

The software error data sets published to date follow no single format. Many software projects gather data to fit the input requirements of some existing error reporting system. The resulting variety in data organization prevents use of any single model in all cases. If data is organized and reported as time between errors (on some time scale consistent with debugging effort), then the Geometric model appears to be the best choice. If data is collected or can be grouped as errors experienced during a series of uniform intervals (uniform with respect to an error's chance of being detected) then Schneidewind's method 3 using the weighted least squares criterion should be best. These conclusions reflect our experience from applying various models to many of the published data sets listed earlier. In those trials these two models consistently gave better and more useful results. At this point we can say little to compare these two models or the respective methods of organizing error data.

While we consider the Schneidewind method 3 and the Geometric models to be the most accurate and useful of the models we have looked at, it is also our contention that these two models need still further validation on an actual system before we can place great faith and confidence in their results. In addition, an estimate of the precision or accuracy of each model needs to be determined. This is where an extension of Forman and Singpurwalla's techniques [37] may prove useful.

3.2 IMPORTANT CONSIDERATIONS IN SOFTWARE ERROR DATA COLLECTION

In designing the requirements for a software error data collection study certain factors have a strong impact on the success or failure of the project.

Here we summarize the aspects which should be carefully reviewed prior to such an investigation.

(1) Proper planning of a project is of utmost importance. The exact requirements must be clearly specified prior to initiation of a data collecting process. We are not recommending any one specific procedure, however the approaches presented on pages 158-59 seem to be adequate for modeling type studies.

(2) The error reporting forms should be designed to be minimal in length, yet include all requirements, and infringe on the programmers (time and effort) as little as possible.

(3) Recording errors by category has been strongly advocated by many researchers in the area of software reliability. This appears to be feasible yet caution must be exercised so that only "true" software errors are documented in such an analysis. Furthermore, during the data collection process, duplicate error reports must be ignored so as to reflect the actual number of errors encountered.

(4) Collecting error data in terms of time-between-errors definitely appears to be the best procedure in that such data can be utilized with models requiring time-between-errors and with those models needing errors per interval. The only potential problem here is the feasibility of collecting data in the aforementioned format.

(5) It is strongly believed that using CPU time as the time reference is the most attractive measure. Calendar time has obvious problems associated with it which were discussed previously in this report.

(6) Finally, when data is being collected, sufficient quantities must be available for model application. Without a good sized data set it becomes difficult to assess the efficiency of the various software reliability models.

3.3 SUGGESTIONS FOR FUTURE STUDY

We feel that the following items are important enough to the field of software reliability to warrant further study and research. We regret that because of insufficient time, lack of the necessary inputs, or less than perfect creativity we could not address these problems ourselves.

(1) It is necessary to have access to the development of a medium to large scale software project at an early enough stage so that software error data can be collected according to the recommendations in Section 3.2. This data will be used with whatever models are desirable (we recommend the Geometric or Schneidewind's method 3, depending on the data format) to make a reliability prediction. Then the software package needs to be monitored for a considerable length of time during the operational phase to collect enough data to produce an ultimate validation of the models' prediction.

(2) When a new module is integrated into the software or when the software is released to the user, there is sometimes a sudden jump in the number of errors discovered. This sudden jump is represented as some kind of bulge in the graph of the failure rate. More needs to be learned about this sudden bulge, namely its shape, height, length, and to where the failure rate returns after the bulge has passed. In other words, what is the relationship between the failure rate before and after such a sudden jump.

(3) One of the points that makes the Schneidewind method 3 model so attractive is its capability of determining and using s, the optimum point at which the earlier data is de-emphasized relative to the most recent data. The strength of the Geometric-Poisson is its simplicity of use and the meaningfulness of its parameters (in fact, Schneidewind method 3 with s=2 and the Geometric-Poisson are numerically equivalent). Both models give a very

good data fit. Therefore it would be very desirable to somehow incorporate the best features of each into a single model.

(4) After a model which requires errors per interval makes prediction below 1 error per interval, the results are difficult to interpret clearly. A more desirable form for the predicitions would be time between errors which steadily increases as the software improves. Therefore, it would be ideal to be able to switch from an errors per interval form to a time between errors form when the failure rate gets very low.

(5) Rudner [119,120] develops some very interesting and promising "tagging" techniques which need to be validated on an actual, controlled test situation with two or more debuggers working on a piece of software.

(6) Forman and Singpurwalla's assurance techniques [37] need to be applied to more models, especially those which predict N, the total number of errors. So far they have only been applied to the Jelinski-Moranda model.

(7) As other software reliability researchers can attest, more good data sets are needed for this kind of research. Efforts need to be made to carefully collect software error data from various projects in sufficient quantity and with sufficient documentation, and these data sets should be made available to those who are exploring software reliability. Rome Air Development Center is presently addressing this issue, and we encourage their efforts.

* * * * * * * * * * * * * * * *

Software reliability continues to be an interesting and challenging area of study. Yet it remains a somewhat illusive attribute and is difficult to determine and predict as precisely as we would like. But this is not to say that it is not a very necessary and significant pursuit. In the development

stage, a knowledge of the reliability of the software would allow for more intelligent decisions as to the allocation of money, time, and effort. In the operational stage, accurate estimates of software reliability not only produce useful and desirable results, but also aid in optimizing the utilization of resources.

APPENDIX A

COMPUTER PROGRAM SOFTW

This appendix contains a listing of a computer program called SOFTW. This program automates the five software reliability models that were initially thought to be the most promising. The specific models are the Jelinski-Moranda model, the extended Jelinski-Moranda model, the Geometric model, the Geometric-Poisson model, and the Schneidewind model (all three methods). The program is written in FORTRAN, and it runs on the CDC 6600 computer at Wright-Patterson AFB, Dayton, Ohio. Both the maximum likelihood estimation, as well as the actual model solution are incorporated in the program.

```
      PROGRAM SOFTW (INPUT, OUTPUT, PASPLT,  DEBUG=OUTPUT, TAPE5=INPUT,      SEPT*77*2   1
     *               TAPE6=OUTPUT, TAPE7=PASPLT)                             SEPT*77*2   2
C                                                                             DRIVER      4
      LOGICAL LERRF,ONLINE                                                    SEPT*77*2   3
      LOGICAL LSUM,LREQ(5)                                                    DRIVER      6
      INTEGER ITIT(24)                                                        SEPT*77*2   4
      INTEGER MODEL(7),IUSE(3,10),INPT(8)                                     ADDSCH2     1
      REAL OBSERV(150)                                                        DRIVER      8
      REAL RESULT(175,3,10),PARMS(3,10),TODATE(150)                           SEPT*77*4   1
      COMMON /PASS/ TODATE,ITIT                                               SEPT*77*2   6
      DATA MAXMOD /7/                                                         ADDSCH2     2
      DATA MAXTAB /10/                                                        DRIVER     13
      DATA MAXOBS /150/                                                       DRIVER     14
      DATA MAXDAT /175/                                                       DRIVER     15
      DATA IUSE /30*0/                                                        DRIVER     16
      DATA ONLINE /.FALSE./                                                   SEPT*77*2   7
      DATA MODEL /"JMEXP","EXTJMEXP","GEOMETRIC","POISSON",                   ADDSCH13    3
     *            "SCHNEID-1","SCHNEID-2","SCHNEID-3"/                        ADDSCH2     3
C                                                                             DRIVER     18
C                                                                             DRIVER     19
C---->   CLEAR/INITIALIZE ALL VARIABLES                                       DRIVER     20
    5 PRINT 980                                                               DRIVER     21
      MAXCOL = 0                                                              DRIVER     22
      ICOLUM = 1                                                              DRIVER     23
      DO 6 I=1,5                                                              DRIVER     24
        LREQ(I) = .FALSE.                                                     DRIVER     25
    6   CONTINUE                                                              DRIVER     26
      LREQ(5) = .TRUE.                                                        DRIVER     27
      DO 355 I=1,10                                                           DRIVER     28
        DO 355 J=1,3                                                          OCT*77*2    1
          DO 355 K=1,175                                                      DRIVER     30
  355       RESULT(K,J,I) = 0.0                                             DRIVER     31
      INPERS = 0                                                              DRIVER     32
      CALL ERRSET (INPERS, 10)                                                DRIVER     33
C                                                                             DRIVER     34
C                                                                             DRIVER     35
   10 CONTINUE                                                                DRIVER     36
      IF (ONLINE)                                  PRINT 790                  SEPT*77*2   8
      READ (5,*) ICOMD,IOPT                                                   DRIVER     37
      STATUS = EOF(5)                                                         DRIVER     38
      IF (STATUS.EQ.0.0)                           GO TO 20                   DRIVER     39
        PRINT 960                                                             DRIVER     40
        GO TO 900                                                             DRIVER     41
   20   CONTINUE                                                             SEPT*77*3   1
C                                                                             DRIVER     43
      IF (ICOMD.NE."INPUT")                        GO TO 120                  DRIVER     44
C---->  "INPUT" IOPT                                                          DRIVER     45
        PRINT 910,ICOMD,IOPT                                                  DRIVER     46
        LREQ(1) = .TRUE.                                                      DRIVER     47
        IF (IOPT.GT.0.AND.IOPT.LE.MAXOBS)  GO TO 105                          DRIVER     48
          PRINT 920,IOPT                                                      DRIVER     49
          LREQ(1) = .FALSE.                                                   DRIVER     50
          GO TO 10                                                           SEPT*77*3   2
  105     CONTINUE                                                            DRIVER     52
        NOBS = IOPT                                                           DRIVER     53
        IF (ONLINE)                                PRINT 790                  SEPT*77*2   9
        READ (5,*) (OBSERV(I),I=1,NOBS)                                       DRIVER     54
```

```
                    PRINT 870,(OBSERV(I),I=1,NOBS)                                  DRIVER     55
                    GO TO 10                                                        DRIVER     56
60          C                                                                       DRIVER     57
              120 IF (ICOMD.NE."USE")                          GO TO 140            DRIVER     58
            C---->  "USE"/"IOPT"                                                    DRIVER     59
                    PRINT 970,ICOMD,IOPT                                            DRIVER     60
                    LREQ(2) = .TRUE.                                                DRIVER     61
65                  DO 125 I=1,MAXMOD                                               DRIVER     62
                         MODNUM = I                                                 DRIVER     63
                         IF (IOPT.EQ.MODEL(I))          GO TO 130                   DRIVER     64
              125        CONTINUE                                                   DRIVER     65
                    PRINT 900,IOPT                                                  DRIVER     66
70                  LREQ(2) = .FALSE.                                               DRIVER     67
              130   CONTINUE                                                        DRIVER     68
                    GO TO 10                                                        DRIVER     69
            C                                                                       DRIVER     70
              140 IF (ICOMD.NE."PREDICT")                      GO TO 160            DRIVER     71
75          C---->  "PREDICT" IOPT                                                  DRIVER     72
                    PRINT 910,ICOMD,IOPT                                            DRIVER     73
                    LREQ(3) = .TRUE.                                                DRIVER     74
                    IF (IOPT.GE.0.AND.NOBS + IOPT.LT.MAXOAT)      GO TO 145         DRIVER     75
                         PRINT 920,IOPT                                             DRIVER     76
80                       LREQ(3) = .FALSE.                                          DRIVER     77
              145        CONTINUE                                                   DRIVER     78
                    IPRED = IOPT                                                    DRIVER     79
                    GO TO 10                                                        DRIVER     80
            C                                                                       DRIVER     81
85            160 IF (ICOMD.NE."COMPUTE")                      GO TO 180            DRIVER     82
            C---->  "COMPUTE" IOPT                                                  DRIVER     83
                    PRINT 910,ICOMD,IOPT                                            DRIVER     84
                    LREQ(4) = .TRUE.                                                DRIVER     85
                    IF (IOPT.GT.0.AND.IOPT+ICOLUM-1.LE.MAXTAB) GO TO 165            DRIVER     86
90                       PRINT 920,IOPT                                             DRIVER     87
                         LREQ(4) = .FALSE.                                          DRIVER     88
                         GO TO 10                                                   SEPT*77*3   3
              165        CONTINUE                                                   DRIVER     90
                    NSOLS = IOPT                                                    DRIVER     91
95                  IDOEND = ICOLUM+NSOLS - 1                                       DRIVER     92
                    IF (IDOEND.GT.MAXCOL)          MAXCOL=IDOEND                    DRIVER     93
                    IF (ONLINE)                                PRINT 790            SEPT*77*2  10
                    READ (5,*) ((IUSE(I,J),I=1,2),J=ICOLUM,IDOEND)                 SEPT*77*2  11
                    PRINT 940,(IUSE(1,I),I=ICOLUM,IDOEND)                           SEPT*77*2  12
100                 PRINT 950,(IUSE(2,I),I=ICOLUM,IDOEND)                           SEPT*77*2  13
                    DO 175 I=ICOLUM,IDOEND                                          DRIVER     97
                         IUSE(3,I) = MODEL(MODNUM)                                  DRIVER     98
                         IF (IUSE(1,I).LT.IUSE(2,I))   GO TO 170                    DRIVER     99
                              PRINT 930,I,IUSE(1,I),IUSE(2,I)                       DRIVER    100
105                           LREQ(4) = .FALSE.                                     DRIVER    101
              170             CONTINUE                                              DRIVER    102
                         DO 173 J=1,2                                               DRIVER    103
                              IF (IUSE(J,I).GT.0.AND.IUSE(J,I).LE.NOBS)  GO TO 172  DRIVER    104
                                   PRINT 920,IUSE(J,I)                              DRIVER    105
110                                LREQ(4) = .FALSE.                                DRIVER    106
              172                  CONTINUE                                         DRIVER    107
              173             CONTINUE                                              DRIVER    108
              175        CONTINUE                                                   DRIVER    109
                    GO TO 300                                                       DRIVER    110
```

```
C
  180 IF(ICOMD.NE."PRINT")                          GO TO 200
C---->    "PRINT"/"IOPT"
          PRINT 970,ICOMD,IOPT
          IPTOPT = 0
          IF (IOPT.EQ."ERRORS".OR.IOPT.EQ."LAMDA")  IPTOPT = 1
          IF (IOPT.EQ."MTTE")                       IPTOPT = 2
          IF (IOPT.EQ."CUMULATIVE")                 IPTOPT = 3
          IF (IPTOPT.NE.0)                          GO TO 185
               PRINT 990,IOPT
               GO TO 10
     185  CONTINUE
C         BEGIN;
               PRINT 840
               PRINT 800,(IUSE(3,I),I=1,MAXCOL)
               PRINT 810,"FROM:",(IUSE(1,I),I=1,MAXCOL)
               PRINT 810,"  TO:",(IUSE(2,I),I=1,MAXCOL)
               PRINT 730
               DO 190 I=1,NOBS
                    TEMP = OBSERV(I)
                    IF (IPTOPT.EQ.3)             TEMP = TODATE(I)
                    PRINT 820,TEMP,(RESULT(I,IPTOPT,J),J=1,MAXCOL)
     190            CONTINUE
               DO 192 I=1,IPRED
                    PRINT 830,(RESULT(I+NOBS,IPTOPT,J),J=1,MAXCOL)
     192            CONTINUE
C              END;
          PRINT 780
          IF (.NOT.ONLINE)                       PRINT 980
          GO TO 10
C
  200 IF (ICOMD.NE."COLUMN")                     GO TO 220
C---->    "COLUMN" IOPT
          PRINT 910,ICOMD,IOPT
          LREQ(5) = .TRUE.
          IF (IOPT.GT.0.AND.IOPT.LE.MAXTAB)      GO TO 205
               PRINT 920,MAXTAB
               LREQ(5) = .FALSE.
               IOPT = 1
     205       CONTINUE
          ICOLUM = IOPT
          MAXCOL = IOPT
          GO TO 10
C
  220 IF (ICOMD.NE."CLEAR")                      GO TO 240
C---->    "CLEAR" "ALL"
          PRINT 970,ICOMD,IOPT
          GO TO 5
C
  240 IF (ICOMD.NE."PLOT")                       GO TO 260
C---->    "PLOT" "IOPT"
          PRINT 970,ICOMD,IOPT
          IF (ONLINE)                            PRINT 790
          READ 890,ITIT
          PRINT 880,ITIT
          IF (IOPT.NE."ERRORS".AND.IOPT.NE."LAMDA")GO TO 242
               WRITE (7) NOBS,IPRED,MAXCOL,IUSE,OBSERV,
```

```
DRIVER   111
DRIVER   112
DRIVER   113
DRIVER   114
SEPT*77*3   4
SEPT*77*3   5
SEPT*77*3   6
SEPT*77*3   7
SEPT*77*3   8
DRIVER   121
DRIVER   122
DRIVER   123
SEPT*77*3   9
SEPT*77*4   3
DRIVER   126
DRIVER   127
DRIVER   128
SEPT*77*2  14
DRIVER   130
SEPT*77*3  10
SEPT*77*3  11
SEPT*77*3  12
DRIVER   132
DRIVER   133
DRIVER   134
DRIVER   135
SEPT*77*3  13
SEPT*77*4   4
SEPT*77*4   5
DRIVER   137
DRIVER   138
DRIVER   139
DRIVER   140
DRIVER   141
DRIVER   142
DRIVER   143
SEPT*77*4   6
DRIVER   144
DRIVER   145
DRIVER   146
DRIVER   147
SEPT*77*4   7
DRIVER   148
DRIVER   149
DRIVER   150
DRIVER   151
DRIVER   152
DRIVER   153
DRIVER   154
SEPT*77*2  16
SEPT*77*2  17
SEPT*77*2  18
SEPT*77*3  16
SEPT*77*2  19
SEPT*77*2  20
SEPT*77*2  21
SEPT*77*2  22
```

```
                     *                ((RESULT(I,1,J),I=1,MAXDAT),J=1,MAXTAB)          SEPT*77*2  23
                     *                ,ITIT                                            SEPT*77*2  24
                          GO TO 10                                                     SEPT*77*2  25
175                242   IF (IOPT.NE."MTTE")                    GO TO 244             SEPT*77*2  26
                          WRITE (7) NOBS,IPRED,MAXCOL,IUSE,OBSERV,                     SEPT*77*2  27
                     *                ((RESULT(I,2,J),I=1,MAXDAT),J=1,MAXTAB)          SEPT*77*2  28
                     *                ,ITIT                                            SEPT*77*2  29
                          GO TO 10                                                     SEPT*77*2  30
180                244   IF (IOPT.NE."CUMULATIVE")              GO TO 246            SEPT*77*2  31
                          WRITE (7) NOBS,IPRED,MAXCOL,IUSE,TODATE,                     SEPT*77*2  32
                     *                ((RESULT(I,3,J),I=1,MAXDAT),J=1,MAXTAB)          SEPT*77*2  33
                     *                ,ITIT                                            SEPT*77*2  34
                          GO TO 10                                                     SEPT*77*2  35
185                246   PRINT 990,IOPT                                               SEPT*77*2  36
                          GO TO 10                                                     SEPT*77*2  37
                C                                                                      SEPT*77*2  38
                  260 IF (ICOMD.NE."ON")                       GO TO 280               SEPT*77*2  39
                C---->    "ON" "LINE"                                                  SEPT*77*2  40
190                       PRINT 970,ICOMD,IOPT                                         SEPT*77*2  41
                          ONLINE = .TRUE.                                              SEPT*77*2  42
                          GO TO 10                                                     SEPT*77*2  43
                C                                                                      SEPT*77*2  44
                  280 IF (ICOMD.NE."END")                      GO TO 290               SEPT*77*2  45
195               C---->    "END" "RUN"                                                SEPT*77*2  46
                          PRINT 970,ICOMD,IOPT                                         SEPT*77*2  47
                          GO TO 500                                                    SEPT*77*2  48
                C                                                                      SEPT*77*2  49
                  290 CONTINUE                                                         SEPT*77*2  50
200               C---->    UNKNOWN KEYWORD                                            DRIVER     156
                       IF (ONLINE)                             GO TO 299               SEPT*77*3  17
                       BACKSPACE 5                                                    DRIVER     157
                       READ (5,890) (INPT(I),I=1,8)                                   DRIVER     158
                       PRINT 980,(INPT(I),I=1,8)                                      DRIVER     159
205                 299 CONTINUE                                                       SEPT*77*3  18
                       PRINT 990,ICOMD                                                DRIVER     160
                       GO TO 10                                                       DRIVER     161
                C                                                                     DRIVER     162
                  300 CONTINUE                                                        DRIVER     163
210               C---->    OUT OF LINE SUBROUTINE ENTERED FROM TAG 175 + 1 LINE       DRIVER     164
                  C---->    TRY TO PROCESS THIS MODEL                                  DRIVER     165
                       LSUM = .TRUE.                                                  DRIVER     166
                       DO 310 I=1,5                                                   DRIVER     167
                          LSUM = LSUM.AND.LREQ(I)                                     DRIVER     168
215                 310   CONTINUE                                                    DRIVER     169
                       IF (LSUM.AND.INPERS.EQ.0)               GO TO 350              DRIVER     170
                          PRINT 860                                                   DRIVER     171
                          GO TO 10                                                    DRIVER     172
                C                                                                     DRIVER     173
220                 350 CONTINUE                                                      DRIVER     174
                  C---->    EXECUTE THE CHOSEN MODEL                                  DRIVER     175
                       PRINT 340                                                      DRIVER     176
                       IDOEND = ICOLUM+NSOLS - 1                                      DRIVER     177
                       DO 400 I=ICOLUM,IDOEND                                         DRIVER     178
225                       IT1 = IUSE(2,I)-IUSE(1,I) + 1                               SEPT*77*2  51
                          IT2 = IUSE(1,I)                                             DRIVER     180
                          GO TO (360,362,364,365,366,367,368) MODNUM                  ADJSCH2      4
                     360        CALL SETJME  (IT1,OBSERV(IT2),PARMS(1,I),LERRF) $ GOTO 370 DRIVER  182
```

```
                362          CALL SETEXTE (IT1,OBSERV(IT2),PARMS(1,I),LERRF) $ GOTO 370 DRIVER    183
230             364          CALL SETJMG  (IT1,OBSERV(IT2),PARMS(1,I),LERRF) $ GOTO 370 DRIVER    184
                365          CALL SETJMGP (IT1,OBSERV(IT2),PARMS(1,I),LERRF) $ GOTO 370 ADDSCH13    6
                366          CALL SETSCH1 (IT1,OBSERV(IT2),PARMS(1,I),LERRF) $ GOTO 370 ADDSCH13    7
                367          CALL SETSCH2 (IUSE(2,I),OBSERV,PARMS(1,I),LERRF,IT2)       ADDSCH2     5
                             GO TO 370                                                  ADDSCH2     6
235             368          CALL SETSCH3 (IUSE(2,I),OBSERV,PARMS(1,I),LERRF,IT2)       ADDSCH2     7
                370      CONTINUE                                                       DRIVER    186
                         DO 354 J=1,3                                                   OCT*77*2    2
                             DO 352 K=1,175                                             OCT*77*2    3
                                 RESULT(K,J,I) = 0.0                                    OCT*77*2    4
240             352              CONTINUE                                               OCT*77*2    5
                354          CONTINUE                                                   OCT*77*2    6
                         IF (.NOT.LERRF)                          GO TO 375             DRIVER    187
                             PRINT 351,MODEL(MODNUM),IUSE(1,I),IUSE(2,I)                DRIVER    188
                             GO TO 400                                                  DRIVER    189
245             375          CONTINUE                                                   DRIVER    190
                             CUMERS = 0.0                                               SEPT*77*3  19
                             DO 377 J=1,NOBS                                            SEPT*77*3  20
                                 CUMERS = CUMERS + OBSERV(J)                            SEPT*77*3  21
                                 TODATE(J) = CUMERS                                     SEPT*77*3  22
250             377              CONTINUE                                               SEPT*77*3  23
                         CUMERS = 0.0                                                   DRIVER    191
                         ACCUM = 0.0                                                    SEPT*77*2  52
                         IDOEND1= NOBS + IPRED - IUSE(1,I) + 1                          DRIVER    192
                         IF (MODNUM.EQ.6.OR.MODNUM.EQ.7)          IDOEND1 = NOBS + IPRED ADDSCH2     8
255             DO 395 J=1,IDOEND1                                                      DRIVER    193
                     IT1 = IUSE(1,I) + J - 1                                            DRIVER    194
                     IF (MODNUM.EQ.6.OR.MODNUM.EQ.7)      IT1 = J                       ADDSCH2     9
                     GO TO (380,382,384,385,386,387,388) MODNUM                         ADDSCH2    10
                360      CALL ESTJME  (J,PARMS(1,I),RESULT(IT1,1,I),                    DRIVER    196
260                  *                  RESULT(IT1,2,I))              $ GOTO 390       SEPT*77*2  53
                382      CALL ESTEXTE (CUMERS,PARMS(1,I),RESULT(IT1,1,I),               DRIVER    196
                     *                  RESULT(IT1,2,I))                               DRIVER    199
                         TEMP = RESULT(IT1,1,I)                                         DEC*77     1
                         IF (J.LE.NOBS)                    TEMP = OBSERV(J)             DEC*77     2
265                      CUMERS = CUMERS + TEMP                                         SEPT*77    3
                         GO TO 390                                                      SEPT*77*2  55
                384      CALL ESTGEOM (J,PARMS(1,I),RESULT(IT1,1,I),                    SEPT*77    4
                     *                  RESULT(IT1,2,I))              $ GOTO 390       SEPT*77*2  56
                385      CALL ESTGEOM (J,PARMS(1,I),RESULT(IT1,1,I),                    ADDSCH13  12
270                  *                  RESULT(IT1,2,I))              $ GOTO 390       ADDSCH13  13
                386      CALL ESTSCH  (J,PARMS(1,I),RESULT(IT1,1,I),                    ADDSCH13  14
                     *                  RESULT(IT1,2,I))              $ GOTO 390       ADDSCH13  15
                387      CALL ESTSCH  (J,PARMS(1,I),RESULT(IT1,1,I),                    ADDSCH13  16
                     *                  RESULT(IT1,2,I))              $ GOTO 390       ADDSCH2   11
275             388      CALL ESTSCH  (J,PARMS(1,I),RESULT(IT1,1,I),                    ADDSCH2   12
                     *                  RESULT(IT1,2,I))                               DRIVER    205
                390      CONTINUE                                                       DRIVER    206
                         ACCUM = ACCUM + RESULT(IT1,1,I)                                SEPT*77*2  57
                         IF (IUSE(1,I).GT.1)          ACCUM = TODATE(IUSE(1,I)-1)       SEPT*77*3  24
280                      RESULT(IT1,3,I) = ACCUM                                        SEPT*77*2  58
                395      CONTINUE                                                       DRIVER    207
                400  CONTINUE                                                           DRIVER    208
                         IDOCUM = IDOEND + 1                                            SEPT*77*2  59
                         GO TO 10                                                       DRIVER    209
285              C                                                                      DRIVER    210
```

```
                C                                                                   DRIVER    211
                C                                                                   DRIVER    212
                  500 CONTINUE                                                      DRIVER    213
                      S T O P                                                       DRIVER    214
      290       C                                                                   DRIVER    215
                C                                                                   DRIVER    216
                  780 FORMAT (1H )                                                  SEPT*77*2  60
                  790 FORMAT (3H ->)                                                SEPT*77*2  61
                  800 FORMAT (1H0,20X,10(1X,A10))                                   DRIVER    218
      295         810 FORMAT (1H ,14X,A5, 10(4X,I2,5X))                             DRIVER    219
                  820 FORMAT (1H ,5X,F11.4,2X,10F11.4)                              DRIVER    220
                  830 FORMAT (1H ,18X,10F11.4)                                      DRIVER    221
                  840 FORMAT (1H0,/)                                                DRIVER    222
                  850 FORMAT (1H0,5X,"***** CAN NOT SOLVE MODEL '",A10,"' ON INTERVAL",  DRIVER  223
      300             *       I6," TO ",I6,/)                                       DRIVER    224
                  860 FORMAT (1H0,5X,"***** REQUIRED DATA MISSING OR IN ERROR *****")  DRIVER  225
                  870 FORMAT (1H ,12X," DATA: ",10F10.2)                            DRIVER    226
                  880 FORMAT (1H ,5X,8A10)                                          DRIVER    227
                  890 FORMAT (8A10)                                                 DRIVER    228
      305         900 FORMAT (1H0,5X,"----> MODEL'",A10,"' NOT KNOWN",/)            DRIVER    229
                  910 FORMAT (1H ,10X,A10,2X,I5)                                    DRIVER    230
                  920 FORMAT (1H0,5X,"----> PARAMETER OUT OF RANGE: ",I10,/)        DRIVER    231
                  930 FORMAT (1H0,5X,"----> ZERO OR NEGATIVE INTERVAL: ",3I10,/)    DRIVER    232
                  940 FORMAT (1H ,12X,"BEGIN: ",3I6)                                DRIVER    233
      310         950 FORMAT (1H ,12X,"  END: ",3I6)                                DRIVER    234
                  960 FORMAT (1H0,5X,"***** END OF INPUT *****",/)                  DRIVER    235
                  970 FORMAT (1H ,10X,A10,2X,A10)                                   DRIVER    236
                  980 FORMAT (1H1)                                                  DRIVER    237
                  990 FORMAT (1H0,5X,"----> KEYWORD '",A10,"' NOT RECOGNIZED",/)    DRIVER    238
      315           E N D                                                           DRIVER    239
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS

5155 SOFTW

VARIABLES		SN	TYPE			RELOCATION				
10203	ACCUM		REAL				10202	CUMERS	REAL	
10162	I		INTEGER				10161	ICOLUM	INTEGER	
10163	ICOMU		INTEGER				10175	IDOEND	INTEGER	
10204	IDOEND1		INTEGER				10165	INPERS	INTEGER	
10237	INPT		INTEGER	ARRAY			10167	IOPT	INTEGER	
10176	IPRED		INTEGER				10176	IPTOPT	INTEGER	
226	ITIT		INTEGER	ARRAY	PASS		10200	IT1	INTEGER	
10201	IT2		INTEGER				10221	IUSE	INTEGER	ARRAY
10166	J		INTEGER				10164	K	INTEGER	
10155	LERRF		LOGICAL				10205	LREQ	LOGICAL	ARRAY
10157	LSUM		LOGICAL				10160	MAXCOL	INTEGER	
7406	MAXDAT		INTEGER				7403	MAXMOD	INTEGER	
7405	MAXOBS		INTEGER				7404	MAXTAB	INTEGER	
10212	MODEL		INTEGER	ARRAY			10172	MODNUM	INTEGER	
10171	NOBS		INTEGER				10174	NSOLS	INTEGER	
10267	OBSERV		REAL	ARRAY			7407	ONLINE	LOGICAL	

VARIABLES	SN	TYPE		RELOCATION					
22717	PARMS	REAL	ARRAY		10515	RESULT	REAL	ARRAY	
10170	STATUS	REAL			10177	TEMP	REAL		
0	TODATE	REAL	ARRAY	PASS					

FILE NAMES		MODE							
2041	DEBUG		0	INPUT	FMT	2041	OUTPUT	FMT	4102 PASPLT
0	TAPE5	MIXED	2041	TAPE6		4102	TAPE7	UNFMT	

EXTERNALS	TYPE	ARGS			
EOF	REAL	1		ERRSET	2
ESTEXTE		4		ESTGEOM	4
ESTJME		4		ESTSCH	4
SETEXTE		4		SETJME	4
SETJMG		4		SETJMGP	4
SETSCH1		4		SETSCH2	5
SETSCH3		5			

STATEMENT LABELS

6155	5		0	6			6213	10	
6226	20		6242	105			6261	120	
0	125		6277	130			6300	140	
6313	145		6315	150			6331	165	
6427	170		6442	172			0	173	
0	175		6450	180			6474	185	
0	190		0	192			6623	200	
6637	205		6642	220			6647	240	
6704	242		6724	244			6744	246	
6747	260		6755	280			6762	290	
6771	299		6774	300			0	310	
7011	350		0	352			0	354	
0	355		7037	360			7046	362	
7053	364		7064	365			7073	366	
7102	367		7110	368			7115	370	
7149	375		0	377			7216	380	
7227	382		7251	384			7262	385	
7273	386		7304	387			7315	388	
7325	390		0	395			7343	400	
7350	500		10002	780	FMT		10004	790	FMT
10006	800	FMT	10011	810	FMT		10015	820	FMT
10021	830	FMT	10024	840	FMT		10026	850	FMT
10037	860	FMT	10046	870	FMT		10052	880	FMT
10060	890	FMT	10057	900	FMT		10065	910	FMT
10070	920	FMT	10076	930	FMT		10104	940	FMT
10110	950	FMT	10114	960	FMT		10121	970	FMT
10124	980	FMT	10126	990	FMT				

LOOPS	LABEL		INDEX	FROM-TO	LENGTH	PROPERTIES	
6104	6		I	25 27	28	INSTACK	
6171	355	*	I	29 32	208	NOT INNER	
6172	355	*	J	30 32	148	NOT INNER	
6201	355		K	31 32	28	INSTACK	
6267	125	*	I	65 68	68	INSTACK EXITS	
6340		*	J	95 98	108	EXT REFS	
6363		*	I	99 99	108	EXT REFS	
6400		*	I	100 100	108	EXT REFS	
6413	175	*	I	101 113	348	EXT REFS NOT INNER	
6430	173	*	J	107 112	158	EXT REFS	

LOOPS	LABEL		INDEX	FROM-TO	LENGTH	PROPERTIES	
6501		*	I	129 129	108	EXT REFS	
6515		*	I	130 130	108	EXT REFS	
6531		*	I	131 131	108	EXT REFS	
6545	190	*	I	133 137	208	EXT REFS	NOT INNER
6665		*	J	136 136	128	EXT REFS	
6674	192	*	I	133 140	228	EXT REFS	NOT INNER
6777		*	J	139 139	138	EXT REFS	
6671		*	J	171 171	118	EXT REFS	
6711		*	J	176 176	118	EXT REFS	
6731		*	J	181 181	118	EXT REFS	
7001	310		I	213 215	38	INSTACK	
7017	400	*	I	224 282	3278	EXT REFS	NOT INNER
7115	394	*	J	237 241	148	NOT INNER	
7129	352		K	236 240	28	INSTACK	
7151	377		J	247 250	48	INSTACK	
7171	395	*	J	255 261	1528	EXT REFS	

COMMON BLOCKS	LENGTH
PASS	174

STATISTICS

PROGRAM LENGTH	146158	6541
BUFFER LENGTH	61448	3172
01 LABELED COMMON LENGTH	2568	174

AD-A069 976 DAYTON UNIV OHIO F/G 9/2
SOFTWARE RELIABILITY: DETERMINATION AND PREDICTION.(U)
JUN 78 L S GEPHART, C M GREENWALD F33615-77-C-3072
UNCLASSIFIED AFFDL-TR-78-77 NL

3 OF 3
AD
A069976

END
DATE
FILMED
7-79
DDC

```
      SUBROUTINE SETJMG (NCOEF, COEF, SAVE, IERR)                          GEOMETRIC   2
      EXTERNAL EVLPOLY,ESTGEOM                                             DEC*77      3
      LOGICAL IERR                                                         GEOMETRIC   4
      INTEGER ORDER                                                        GEOMETRIC   5
      REAL SAVE(2),COEF(NCOEF)                                             GEOMETRIC   6
      COMMON /PASS/ MODEL,ORDER,AT(50)                                     GEOMETRIC   7
C                                                                          GEOMETRIC   8
C--->    ACCEPTS 'NCOEF' OBSERVATIONS OF TIME BETWEEN SOFTWARE             GEOMETRIC   9
C        ERRORS IN ROW 'COEF' COMPUTES THE PARAMETERS OF THE               GEOMETRIC  10
C        JELINSKI-MORANDA GEOMETRIC MODEL, AND RETURNS THE                 GEOMETRIC  11
C        PARAMETER VALUES IN 'SAVE'.  'SAVE(1)' = ESTIMATE OF 'K'          GEOMETRIC  12
C                                     'SAVE(2)' = ESTIMATE OF 'D'          GEOMETRIC  13
C        THE MODEL IS   Z(I)(T) = D*K**(I-1.)                              GEOMETRIC  14
C                                                                          GEOMETRIC  15
C                                                                          GEOMETRIC  16
      IERR = .FALSE.                                                       GEOMETRIC  17
      MODEL = "POLYNOMIAL"                                                 GEOMETRIC  18
C                                                                          GEOMETRIC  19
      DO 10 I=1,NCOEF                                                      GEOMETRIC  20
         AT(I) = (I-(NCOEF+1)/2.0) * COEF(I)                               GEOMETRIC  21
   10    CONTINUE                                                          GEOMETRIC  22
C                                                                          GEOMETRIC  23
      ORDER = NCOEF-1                                                      GEOMETRIC  24
C                                                                          GEOMETRIC  25
      ALOW = 0.0                                                           GEOMETRIC  26
      AHIGH = 1.0                                                          GEOMETRIC  27
      IF (MSIGN(EVLPOLY(ALOW)).NE.MSIGN(EVLPOLY(AHIGH)))     GO TO 20      GEOMETRIC  28
         PRINT *,"SETJMG  -- SIGN(P(0)) = SIGN(P(1))",                     GEOMETRIC  29
     *                     EVLPOLY(ALOW),EVLPOLY(AHIGH)                    GEOMETRIC  30
   20    CONTINUE                                                          GEOMETRIC  31
      EPS = 1.0E-6                                                         GEOMETRIC  32
      NSIG = 8                                                             GEOMETRIC  33
      MAXFN = 100                                                          GEOMETRIC  34
      CALL ZBRENT (EVLPOLY,EPS,NSIG ,ALOW,AHIGH,MAXFN,IER)                 GEOMETRIC  35
      IF (IER.EQ.0)          GO TO 25                                      GEOMETRIC  36
         IERR = .TRUE.                                                     GEOMETRIC  37
         PRINT *,"IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER              GEOMETRIC  38
     *                  ," TO SETJMG."                                     GEOMETRIC  39
         R E T U R N                                                       GEOMETRIC  40
   25    CONTINUE                                                          GEOMETRIC  41
      HATK = AHIGH                                                         GEOMETRIC  42
C                                                                          GEOMETRIC  43
C--->    EVALUATE  SUM(T(I) * K**(I-1))  FOR  I=1,NCOEF                    GEOMETRIC  44
C                                                                          GEOMETRIC  45
      VAL = COEF(NCOEF)                                                    GEOMETRIC  46
      DO 30 I=1,ORDER                                                      GEOMETRIC  47
         VAL = HATK*VAL + COEF(NCOEF-I)                                    GEOMETRIC  48
   30    CONTINUE                                                          GEOMETRIC  49
C                                                                          GEOMETRIC  50
      HATD = FLOAT(NCOEF) / VAL                                            GEOMETRIC  51
C                                                                          GEOMETRIC  52
      SAVE(1) = HATK                                                       OCT*77*2    7
      SAVE(2) = HATD                                                       OCT*77*2    8
      PRINT *," GEOMETRIC PARMS: K HAT = ",HATK,"  D HAT = ",HATD          GEOMETRIC  53
      CALL SUMSQR (ESTGEOM,NCOEF,COEF,SAVE,2,0.0)                          OCT*77*2    9
      PRINT *,"                    WITH ",MAXFN," EVALUATIONS."            GEOMETRIC  54
      R E T U R N                                                          GEOMETRIC  57
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 SETUMG

VARIABLES	SN	TYPE			RELOCATION					
205	AHIGH	REAL				205	ALOW	REAL		
2	AT	REAL	ARRAY	PASS		0	COEF	REAL	ARRAY	F.P.
207	EPS	REAL				215	HATO	REAL		
216	HATK	REAL				204	I	INTEGER		
212	IER	INTEGER				0	IERR	LOGICAL		F.P.
211	MAXFN	INTEGER				0	MODEL	INTEGER		PASS
0	NCOEF	INTEGER		F.P.		210	NSIG	INTEGER		
1	ORDER	INTEGER		PASS		0	SAVE	REAL	ARRAY	F.P.
214	VAL	REAL								

FILE NAMES	MODE
OUTPUT	FREE

EXTERNALS	TYPE	ARGS			
CSTGEOM		0		EVLPOLY	REAL 1
MSIGN	INTEGER	1		SUMSQWE	6
ZBRENT		7			

INLINE FUNCTIONS	TYPE	ARGS	
FLOAT	REAL	1	INTRIN

STATEMENT LABELS

0	10	46	20	60	25		
0	30						

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
16	10	I	19 21	48	INSTACK
72	30	I	46 48	48	INSTACK

COMMON BLOCKS	LENGTH
PASS	52

STATISTICS

PROGRAM LENGTH	2508	168
CM LABELED COMMON LENGTH	648	52

181

```
      SUBROUTINE SETJMGP (NOBS,OBSERV,SAVE,ERROR)                       POISSON    2
      EXTERNAL EVLPOLY,ESTGEOM                                          SEPT*77*4  8
      LOGICAL ERROR                                                     POISSON    4
      INTEGER ORDER                                                     POISSON    5
      REAL SAVE(2),OBSERV(NOBS)                                         POISSON    6
      COMMON /PASS/ MODEL,ORDER,AT(50)                                  POISSON    7
C                                                                       POISSON    8
C----->   ACCEPTS 'NOBS' OBSERVATIONS OF ERRORS PER INTERVAL IN         POISSON    9
C         ROW 'OBSERV', COMPUTES THE PARAMETERS OF THE JELINSKI-        POISSON   10
C         MORANDA GEOMETRIC POISSON MODEL, AND RETURNS THE              POISSON   11
C         PARAMETER VALUES IN 'SAVE'.  'SAVE(1)' = 'K'                  POISSON   12
C                                      'SAVE(2)' = 'LAMDA'              POISSON   13
C         THE MODEL IS Z(T)(I) = LAMDA*K**(I-1)                         POISSON   14
C                                                                       POISSON   15
      ERROR = .FALSE.                                                   POISSON   16
      MODEL = "POLYNOMIAL"                                              POISSON   17
C                                                                       POISSON   18
      SUM1 = OBSERV(1)                                                  POISSON   19
      SUM2 = 0.0                                                        POISSON   20
      DO 10 I=2,NOBS                                                    POISSON   21
         SUM1 = SUM1 + OBSERV(I)                                        POISSON   22
         SUM2 = SUM2 + OBSERV(I)*(I-1)                                  POISSON   23
   10    CONTINUE                                                       POISSON   24
      BIGP = SUM1/SUM2                                                  POISSON   25
C                                                                       POISSON   26
      ORDER = NOBS+1                                                    POISSON   27
      NCOEF = ORDER + 1                                                 POISSON   28
      DO 15 I=1,NCOEF                                                   POISSON   29
         AT(I) = 0.0                                                    POISSON   30
   15    CONTINUE                                                       POISSON   31
      AT(NCOEF) = 1.0 + BIGP*(1-NOBS)                                   POISSON   32
      AT(NCOEF-1) = NOBS*BIGP - 1.0                                     POISSON   33
      AT(2) = -(BIGP + 1.0)                                             DEC*77     4
      AT(1) = 1.0                                                       POISSON   35
C                                                                       POISSON   36
      ALOW = 0.0                                                        OCT*77*1   1
      CALL FINDCRS (EVLPOLY, MSIGN(EVLPOLY(ALOW)), AHIGH, .9, .9)       OCT*77*1   2
      EPS = 1.0E-50                                                     OCT*77*1   3
      NSIG = 15                                                         OCT*77*1   4
      MAXFN = 100                                                       POISSON   45
      CALL ZBRENT (EVLPOLY,EPS,NSIG ,ALOW,AHIGH,MAXFN,IER)              POISSON   46
      IF (IER.EQ.0)           GO TO 25                                  POISSON   47
         ERROR = .TRUE.                                                 POISSON   48
         PRINT *,"IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER           POISSON   49
     *                    ," TO SETJMGP."                              POISSON   50
         R E T U R N                                                    POISSON   51
   25    CONTINUE                                                       POISSON   52
      HATK = AHIGH                                                      POISSON   53
C                                                                       POISSON   54
      PROD = 1.0                                                        POISSON   55
      SUMP = 0.0                                                        POISSON   56
      DO 30 I=1,NOBS                                                    POISSON   57
         SUMP = SUMP + PROD                                             POISSON   58
         PROD = PROD * HATK                                             POISSON   59
   30    CONTINUE                                                       POISSON   60
      HLAMDA = SUM1/SUMP                                                POISSON   61
C                                                                       POISSON   62
```

```
                      SAVE(1) = HATK                                                    SEPT*77*4   9
                      SAVE(2) = HLAMDA                                                  SEPT*77*4   10
   60         C                                                                         SEPT*77*4   11
                      PRINT *," POISSON PARAMS: K HAT = ",HATK," LAMDA = ",HLAMDA       POISSON     63
                      CALL SUMSQWE (ESTGEOM,NOBS,OBSERV,SAVE,1,0.0)                     OCT*77*2    10
                      PRINT *,"                    WITH ",MAXFN," EVALUATIONS."         POISSON     64
                      R E T U R N                                                       POISSON     67
   65                   E N D                                                          POISSON     68
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 SETJMGP

VARIABLES	SN	TYPE		RELOCATION					
210	AHIGH	REAL			207	ALOW	REAL		
2	AT	REAL	ARRAY	PASS	205	BIGP	REAL		
211	EPS	REAL			0	ERROR	LOGICAL		F.P.
219	HATK	REAL			220	HLAMDA	REAL		
204	I	INTEGER			214	IER	INTEGER		
213	MAXFN	INTEGER			0	MODEL	INTEGER		PASS
206	NODEF	INTEGER			0	NOBS	INTEGER		F.P.
212	NSIG	INTEGER			0	OBSERV	REAL	ARRAY	F.P.
1	ORDER	INTEGER		PASS	216	PROD	REAL		
3	SAVE	REAL	ARRAY	F.P.	217	SUMP	REAL		
202	SUM1	REAL			203	SUM2	REAL		

FILE NAMES	MODE
OUTPUT	FREE

EXTERNALS	TYPE	ARGS			
ESTGEOM		0	EVLPOLY	REAL	1
FINDCRS		5	MSIGN	INTEGER	1
SUMSQWE		6	ZBRENT		7

STATEMENT LABELS
0 10 0 15 70 25
0 30

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
17	10	I	20 23	68	INSTACK
34	15	I	28 30	28	INSTACK
77	30	I	52 55	43	INSTACK

COMMON BLOCKS	LENGTH
PASS	52

STATISTICS

PROGRAM LENGTH	2468	166
CM LABELED COMMON LENGTH	648	52

```
      1                 FUNCTION EVLPOLY (X)                                          POISSON    69
                        INTEGER ORDER                                                 POISSON    70
                        COMMON /PASS/ MODEL,ORDER,COEF(50)                            POISSON    71
                  C                                                                   POISSON    72
      5           C--->    EVALUATE A POLYNOMIAL OF ORDER 'ORDER' WITH ITS           POISSON    73
                  C        COEFFICIENTS IN 'COEF' IN ORDER OF INCREASING EXPONENTS.  POISSON    74
                  C                                                                   POISSON    75
                        IF (MODEL.NE."POLYNOMIAL")       PRINT *,"***** BAD CALL"     POISSON    76
                     *                                        ," TO EVLPOLY *****"    POISSON    77
     10           C                                                                   POISSON    78
                        SUM = COEF(ORDER+1)                                           POISSON    79
                        DO 10 I=1,ORDER                                               POISSON    80
                           SUM = SUM*X + COEF(ORDER-I+1)                              POISSON    81
                     10    CONTINUE                                                   POISSON    82
     15                 EVLPOLY = SUM                                                 POISSON    83
                        R E T U R N                                                   POISSON    84
                           E N D                                                      POISSON    85
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
4 EVLPOLY

VARIABLES	SN	TYPE			RELOCATION				
2	COEF	REAL	ARRAY	PASS	34	EVLPOLY	REAL		
30	I	INTEGER			0	MODEL	INTEGER	PASS	
1	ORDER	INTEGER		PASS	35	SUM	REAL		
0	X	REAL		F.P.					

FILE NAMES	MODE
OUTPUT	FREE

STATEMENT LABELS
0 10

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
21	10	I	12 14	38	INSTACK

COMMON BLOCKS	LENGTH
PASS	52

STATISTICS

PROGRAM LENGTH	458	37
CM LABELED COMMON LENGTH	648	52

```
      1                 SUBROUTINE ESTGEOM(I,SAVE,LAMDA,MTTE)                    POISSON   86
                        REAL LAMDA,MTTE,SAVE(2)                                  POISSON   87
                  C                                                              POISSON   88
                  C---->    ACCEPTS 'I' NUMBEROF ERRORS TO DATE, 'SAVE' A ROW OF POISSON   89
      5           C         PARAMETERS COMPUTED BY SETJMG, AND APPLYS THE JELINSKI-MORANDS POISSON   90
                  C         GEOMETRIC MODEL TO GIVE 'LAMDA' A ESTIMATE OF THE    POISSON   91
                  C         CURRENT ERROR RATE, AND 'MTTE' AN ESTIMATE OF TIME TO POISSON   92
                  C         NEXT ERROR                                           POISSON   93
                  C                                                              POISSON   94
     10                   HATK = SAVE(1)                                         POISSON   95
                          HATO = SAVE(2)                                         POISSON   96
                          LAMDA = HATO * HATK**(I-1)                             POISSON   97
                          MTTE = 1.0 / LAMDA                                     POISSON   98
                          R E T U R N                                            POISSON   99
     15                     E N D                                               POISSON  100
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 ESTGEOM

VARIABLES	SN	TYPE	RELOCATION					
20	HATO	REAL		25	HATK	REAL		
0	I	INTEGER	F.P.	0	LAMDA	REAL		F.P.
0	MTTE	REAL	F.P.	0	SAVE	REAL	ARRAY	F.P.

STATISTICS
PROGRAM LENGTH 278 23

```
      SUBROUTINE SETSCH1 (NOBS, OBSERV, SAVE, ERROR)                    SCHNEID-1   2
      EXTERNAL EVLPOLY                                                  SCHNEID-1   3
      LOGICAL ERROR                                                     SCHNEID-1   4
      INTEGER ORDER                                                     SCHNEID-1   5
      REAL SAVE(2),OBSERV(NOBS)                                         SCHNEID-1   6
      COMMON /PASS/ MODEL,ORDER,AT(50)                                  SCHNEID-1   7
C                                                                       SCHNEID-1   8
C---->   ACCEPTS 'NOBS' OBSERVATIONS OF ERRORS PER INTERVAL IN          SCHNEID-1   9
C        ROW 'OBSERV', COMPUTES THE PARAMETERS OF THE SCHNEIDEWIND      SCHNEID-1  10
C        ANALYTICAL (NUMBER ONE) MODEL, AND RETURNS THE                 SCHNEID-1  11
C        PARAMETER VALUES IN 'SAVE'.  'SAVE(1)' = 'BETA'                SCHNEID-1  12
C                                     'SAVE(2)' = 'ALPHA'               SCHNEID-1  13
C        THE MODEL IS Z(I)(T) = ALPHA*EXP(-BETA*I)                      SCHNEID-1  14
C                    M(I) = (ALPHA/BETA) *                              SCHNEID-1  15
C                           (EXP(-BETA*(I-1)) - EXP(-BETA*I))           SCHNEID-1  16
C                                                                       SCHNEID-1  17
      ERROR = .FALSE.                                                   SCHNEID-1  18
      MODEL = "POLYNOMIAL"                                              SCHNEID-1  19
C                                                                       SCHNEID-1  20
      SUM1 = OBSERV(1)                                                  SCHNEID-1  21
      SUM2 = 0.0                                                        SCHNEID-1  22
      DO 10 I=2,NOBS                                                    SCHNEID-1  23
         SUM1 = SUM1 + OBSERV(I)                                        SCHNEID-1  24
         SUM2 = SUM2 + OBSERV(I)*(I-1)                                  SCHNEID-1  25
   10    CONTINUE                                                       SCHNEID-1  26
      BIGA = SUM2/SUM1                                                  SCHNEID-1  27
C                                                                       SCHNEID-1  28
      ORDER = NOBS+1                                                    SCHNEID-1  29
      NCOEF = ORDER + 1                                                 SCHNEID-1  30
      DO 15 I=1,NCOEF                                                   SCHNEID-1  31
         AT(I) = 0.0                                                    SCHNEID-1  32
   15    CONTINUE                                                       SCHNEID-1  33
      AT(NCOEF) = BIGA                                                  SCHNEID-1  34
      AT(NCOEF-1) = -(BIGA + 1.0)                                       SCHNEID-1  35
      AT(2) = NOBS - BIGA                                               SCHNEID-1  36
      AT(1) = BIGA + 1.0 - NOBS                                         SCHNEID-1  37
C                                                                       SCHNEID-1  38
      ALOW = 1.1                                                        SCHNEID-1  39
      AHIGH = 2.71828                                                   SCHNEID-1  40
      IF (MSIGN(EVLPOLY(ALOW)).NE.MSIGN(EVLPOLY(AHIGH)))    GO TO 20    SCHNEID-1  41
         PRINT *,"SETSCH1-- SIGN(P(",ALOW,")) = SIGN(P(",AHIGH,         SCHNEID-1  42
     *                    "))    ",EVLPOLY(ALOW),"   ",EVLPOLY(AHIGH)  SCHNEID-1  43
   20    CONTINUE                                                       SCHNEID-1  44
      EPS = 1.0E-6                                                      SCHNEID-1  45
      NSIG = 8                                                          SCHNEID-1  46
      MAXFN = 100                                                       SCHNEID-1  47
      CALL ZBRENT (EVLPOLY,EPS,NSIG ,ALOW,AHIGH,MAXFN,IER)              SCHNEID-1  48
      IF (IER.EQ.0)           GO TO 25                                  SCHNEID-1  49
         ERROR = .TRUE.                                                 SCHNEID-1  50
         PRINT *,"IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER           SCHNEID-1  51
     *                    ," TO SETSCH1."                              SCHNEID-1  52
         R E T U R N                                                    SCHNEID-1  53
   25    CONTINUE                                                       SCHNEID-1  54
      BETA = ALOG(AHIGH)                                                SCHNEID-1  55
C                                                                       SCHNEID-1  56
      ALPHA = (SUM1*BETA)/(1.0-EXP(-BETA*NOBS))                         SCHNEID-1  57
C                                                                       SCHNEID-1  58
```

```
                      PRINT *,"  SCHNEIDEWIND (1) PARMS: ALPHA = ",ALPHA,                SCHNEID-1  59
                     *       "  BETA = ",BETA                                            SCHNEID-1  60
     60               PRINT *,"                          WITH ",MAXFN," EVALUATIONS."     SCHNEID-1  61
                      SAVE(1) = BETA                                                      SCHNEID-1  62
                      SAVE(2) = ALPHA                                                     SCHNEID-1  63
                      R E T U R N                                                         SCHNEID-1  64
                        E N D                                                             SCHNEID-1  65


        SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
     3  SETSCH1

VARIABLES     SN  TYPE          RELOCATION                 SN  TYPE          RELOCATION
   212  AHIGH     REAL                                   211  ALOW      REAL
   220  ALPHA     REAL                                     2  AT        REAL        ARRAY   PASS
   217  BETA      REAL                                   207  BIGA      REAL
   213  EPS       REAL                                     0  ERROR     LOGICAL             F.P.
   206  I         INTEGER                                216  IER       INTEGER
   215  MAXFN     INTEGER                                  0  MODEL     INTEGER             PASS
   210  NCOEF     INTEGER                                  0  NOBS      INTEGER             F.P.
   214  NSIG      INTEGER                                  0  OBSERV    REAL        ARRAY   F.P.
     1  ORDER     INTEGER          PASS                    0  SAVE      REAL        ARRAY   F.P.
   204  SUM1      REAL                                   205  SUM2      REAL

FILE NAMES      MODE
     OUTPUT     FREE

EXTERNALS       TYPE   ARGS
     ALOG       REAL     1 LIBRARY                        EVLPOLY   REAL      1
     EXP        REAL     1 LIBRARY                        MSIGN     INTEGER   1
     ZBRENT              7

STATEMENT LABELS
     0  10                                    0  15                                  67  20
   101  25

LOOPS  LABEL  INDEX     FROM-TO     LENGTH     PROPERTIES
    17  10    I           22 25         68     INSTACK
    34  15    I           30 32         28     INSTACK

COMMON BLOCKS   LENGTH
     PASS          52

STATISTICS
   PROGRAM LENGTH                  2628     178
   CM LABELED COMMON LENGTH         048      52
```

```
      1                SUBROUTINE ESTSCH  (I, SAVE, LAMDA, MTTE)                    SCHNEID-1  66
                       REAL LAMDA, MTTE, SAVE(2)                                    SCHNEID-1  67
                     C                                                              SCHNEID-1  68
                     C--->    ACCEPTS 'I' NUMBER OF ERRORS TO DATE, 'SAVE' A ROW OF SCHNEID-1  69
      5              C        PARAMETERS COMPUTED BY SETSCHN AND APPLYS THE SCHNEIDEWIND SCHNEID-1  70
                     C        ANALYTICAL MODEL 1,2,3 TO GIVE 'LAMDA' AN ESTIMATE OF THE SCHNEID-1  71
                     C        CURRENT ERROR RATE, AND 'MTTE' AN ESTIMATE OF TIME TO     SCHNEID-1  72
                     C        NEXT ERROR                                                SCHNEID-1  73
                     C                                                                  SCHNEID-1  74
     10                      BETA = SAVE(1)                                             SCHNEID-1  75
                             ALPHA = SAVE(2)                                            SCHNEID-1  76
                             LAMDA = (ALPHA/BETA) * (EXP(-BETA*(I-1)) - EXP(-BETA*I))   SCHNEID-1  77
                             MTTE = 1.0 / LAMDA                                         SCHNEID-1  78
                             R E T U R N                                                SCHNEID-1  79
     15                        E N D                                                   SCHNEID-1  80
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 ESTSCH

VARIABLES	SN	TYPE	RELOCATION					
41	ALPHA	REAL		40	BETA	REAL		
0	I	INTEGER	F.P.	0	LAMDA	REAL		F.P.
0	MTTE	REAL	F.P.	0	SAVE	REAL	ARRAY	F.P.

EXTERNALS	TYPE	ARGS
EXP	REAL	1 LIBRARY

STATISTICS
PROGRAM LENGTH 428 34

```
      SUBROUTINE SETSCH2 (NOBS, OBSERV, SAVE, ERROR, NIG)                    SCHNEID-2   2
      EXTERNAL EVLPOLY,ESTSCH                                                SCHNEID-2   3
      LOGICAL ERROR                                                          SCHNEID-2   4
      INTEGER ORDER                                                          SCHNEID-2   5
      REAL SAVE(2),OBSERV(NOBS)                                              SCHNEID-2   6
      COMMON /PASS/ MODEL,ORDER,AT(50)                                       SCHNEID-2   7
C                                                                            SCHNEID-2   8
C---->  ACCEPTS 'NOBS' OBSERVATIONS OF ERRORS PER INTERVAL IN                 SCHNEID-2   9
C       ROW 'OBSERV', COMPUTES THE PARAMETERS OF THE SCHNEIDEWIND            SCHNEID-2  10
C       ANALYTICAL (NUMBER THREE) MODEL, AND RETURNS THE                     SCHNEID-2  11
C       PARAMETER VALUES IN 'SAVE'.  'SAVE(1)' = 'BETA'                      SCHNEID-2  12
C                                    'SAVE(2)' = 'ALPHA'                     SCHNEID-2  13
C       THE MODEL IS Z(I)(T) = ALPHA*EXP(-BETA*I)                            SCHNEID-2  14
C                      M(I) = (ALPHA/BETA) *                                 SCHNEID-2  15
C                             (EXP(-BETA*(I-1)) - EXP(-BETA*I))              SCHNEID-2  16
C                                                                            SCHNEID-2  17
      IF (NIG.GE.2.AND.NIG.LT.NOBS)                GO TO 5                   SCHNEID-2  18
      PRINT *," SETSCH2 --'S' PARAMETER MUST BE 2.LE.S.LT.NOBS --"           SCHNEID-2  19
      ERROR = .TRUE.                                                         SCHNEID-2  20
      RETURN                                                                 SCHNEID-2  21
    5 CONTINUE                                                               SCHNEID-2  22
C                                                                            SCHNEID-2  23
      ERROR = .FALSE.                                                        SCHNEID-2  24
      MODEL = "POLYNOMIAL"                                                   SCHNEID-2  25
C                                                                            SCHNEID-2  26
      SUM1 = OBSERV(1)                                                       SCHNEID-2  27
      DO 10 I=2,NOBS                                                         SCHNEID-2  28
         SUM1 = SUM1 + OBSERV(I)                                             SCHNEID-2  29
   10 CONTINUE                                                               SCHNEID-2  30
      SUM2 = 0.0                                                             SCHNEID-2  31
      SUM3 = OBSERV(NIG)                                                     SCHNEID-2  32
      IDOEND = NOBS - NIG                                                    SCHNEID-2  33
      DO 12 I=1,IDOEND                                                       SCHNEID-2  34
         SUM2 = SUM2 + OBSERV(NIG+I)*I                                       SCHNEID-2  35
         SUM3 = SUM3 + OBSERV(NIG+I)                                         SCHNEID-2  36
   12 CONTINUE                                                               SCHNEID-2  37
      BIGA = SUM2/SUM3                                                       SCHNEID-2  38
C                                                                            SCHNEID-2  39
      BIGC = NOBS - NIG + 1                                                  SCHNEID-2  40
      ORDER = NOBS - NIG + 2                                                 SCHNEID-2  41
      NCOEF = ORDER + 1                                                      SCHNEID-2  42
      DO 15 I=1,NCOEF                                                        SCHNEID-2  43
         AT(I) = 0.0                                                         SCHNEID-2  44
   15 CONTINUE                                                               SCHNEID-2  45
      AT(NCOEF) = BIGA                                                       SCHNEID-2  46
      AT(NCOEF-1) = -(BIGA + 1.0)                                            SCHNEID-2  47
      AT(2) = BIGC - BIGA                                                    SCHNEID-2  48
      AT(1) = BIGA + 1.0 - BIGC                                              SCHNEID-2  49
C                                                                            SCHNEID-2  50
      ALOW = 1.1                                                             SCHNEID-2  51
      AHIGH = 2.71828                                                        SCHNEID-2  52
      IF (MSIGN(EVLPOLY(ALOW)).NE.MSIGN(EVLPOLY(AHIGH)))    GO TO 20         SCHNEID-2  53
         ERROR = .TRUE.                                                      SCHNEID-2  54
         PRINT *,"SETSCH2-- SIGN(P(",ALOW,")) = SIGN(P(",AHIGH,              SCHNEID-2  55
     *                    ")) ",EVLPOLY(ALOW),"  ",EVLPOLY(AHIGH)           SCHNEID-2  56
      PRINT 900,((EVLPOLY(FLOAT(I+J-11)/100.0),I=1,10),J=110,280,10)         SCHNEID-2  57
  900 FORMAT (1H ,10F13.6)                                                   SCHNEID-2  58
```

```
                      RETURN                                                    SCHNEID-2  59
                   20 CONTINUE                                                  SCHNEID-2  60
      60              EPS = 1.0E-6                                              SCHNEID-2  61
                      NSIG = 8                                                  SCHNEID-2  62
                      MAXFN = 100                                               SCHNEID-2  63
                      CALL ZBRENT (EVLPOLY,EPS,NSIG ,ALOW,AHIGH,MAXFN,IER)      SCHNEID-2  64
                      IF (IER.EQ.0)          GO TO 25                           SCHNEID-2  65
      65                 ERROR = .TRUE.                                         SCHNEID-2  66
                         PRINT *,"IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER   SCHNEID-2  67
                     *                    ," TO SETSCH2."                       SCHNEID-2  68
                         RETURN                                                 SCHNEID-2  69
                   25 CONTINUE                                                  SCHNEID-2  70
      70              BETA = ALOG(AHIGH)                                        SCHNEID-2  71
                   C                                                            SCHNEID-2  72
                      ALPHA = (SUM1*BETA)/(1.0-EXP(-BETA*NOBS))                 SCHNEID-2  73
                   C                                                            SCHNEID-2  74
                      SAVE(1) = BETA                                            SCHNEID-2  75
      75              SAVE(2) = ALPHA                                           SCHNEID-2  76
                   C                                                            SCHNEID-2  77
                      PRINT *,"  SCHNEIDEWIND (2) PARMS: ALPHA = ",ALPHA,       SCHNEID-2  81
                     *        "  BETA = ",BETA                                  OCT*77*2   11
                      CALL SUMSQRE (ESTSCH,NOBS,OBSERV,SAVE,1,0.0)              OCT*77*2   12
      80              CALL SUMSQRE (ESTSCH,NOBS,OBSERV,SAVE,1,BETA)             OCT*77*2   13
                      PRINT *,"                          WITH ",MAXFN," EVALUATIONS."  SCHNEID-2  83
                   C                                                            SCHNEID-2  84
                      RETURN                                                    SCHNEID-2  85
                      END                                                       SCHNEID-2  86
```

190

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
5 SETSCH2

VARIABLES	SN	TYPE	RELOCATION						
335	AHIGH	REAL		334	ALOW	REAL			
344	ALPHA	REAL		2	AT	REAL	ARRAY	PASS	
343	BETA	REAL		331	BIGA	REAL			
332	BIGC	REAL		337	EPS	REAL			
0	ERROR	LOGICAL	F.P.	325	I	INTEGER			
330	IDGEND	INTEGER		342	IER	INTEGER			
336	J	INTEGER		341	MAXFN	INTEGER			
0	MODEL	INTEGER	PASS	333	NCOEF	INTEGER			
0	NIS	INTEGER	F.P.	0	NOBS	INTEGER		F.P.	
340	NSIG	INTEGER		0	OBSERV	REAL	ARRAY	F.P.	
1	ORDER	INTEGER	PASS	0	SAVE	REAL	ARRAY	F.P.	
324	SUM1	REAL		326	SUM2	REAL			
327	SUM3	REAL							

FILE NAMES	MODE
OUTPUT	MIXED

```
EXTERNALS          TYPE    ARGS
      ALOG         REAL       1 LIBRARY          ESTSCH                    0
      EVLPOLY      REAL       1                  EXP          REAL         1 LIBRARY
      ISIGN        INTEGER    1                  SUMSQWE                   6
      ZBRENT                  7

INLINE FUNCTIONS   TYPE    ARGS
      FLOAT        REAL       1  INTRIN

STATEMENT LABELS
     14  5                                  0  10                          0  12
      0  15                               141  20                        153  25
    270  900      FMT

LOOPS  LABEL     INDEX        FROM-TO     LENGTH     PROPERTIES
     2+  10      I             27 29         38      INSTACK
     43  12      I             33 36         58      INSTACK
     62  15      I             42 44         28      INSTACK
    121          * J           55 56        108                EXT REFS  NOT INNER
    122          * I           55 56        138                EXT REFS

COMMON BLOCKS    LENGTH
      PASS           52

STATISTICS
  PROGRAM LENGTH                4148      268
  CM LABELED COMMON LENGTH       64B       52
```

```
      SUBROUTINE SETSCH3 (NOBS, OBSERV, SAVE, ERROR, NAVG)                SCHNEID-3   2
      EXTERNAL EVLPOLY,ESTSCH                                             SEPT*77*4  15
      LOGICAL ERROR                                                       SCHNEID-3   4
      INTEGER ORDER                                                       SCHNEID-3   5
      REAL SAVE(2),OBSERV(NOBS)                                           SCHNEID-3   6
      COMMON /PASS/ MODEL,ORDER,AT(50)                                    SCHNEID-3   7
C                                                                         SCHNEID-3   8
C---->   ACCEPTS 'NOBS' OBSERVATIONS OF ERRORS PER INTERVAL IN            SCHNEID-3   9
C        ROW 'OBSERV', COMPUTES THE PARAMETERS OF THE SCHNEIDEWIND        SCHNEID-3  10
C        ANALYTICAL (NUMBER THREE) MODEL, AND RETURNS THE                 SCHNEID-3  11
C        PARAMETER VALUES IN 'SAVE'.  'SAVE(1)' = 'BETA'                  SCHNEID-3  12
C                                     'SAVE(2)' = 'ALPHA'                 SCHNEID-3  13
C        THE MODEL IS Z(I)(T) = ALPHA*EXP(-BETA*I)                        SCHNEID-3  14
C                     M(I) = (ALPHA/BETA) *                               SCHNEID-3  15
C                            (EXP(-BETA*(I-1)) - EXP(-BETA*I))            SCHNEID-3  16
C                                                                         SCHNEID-3  17
      IF (NAVG.GE.2.AND.NAVG.LT.NOBS)              GO TO 5                SCHNEID-3  18
         PRINT *," SETSCH3 -- 'S' PARAMETER MUST BE 2.LE.S.LT.NOBS --"    SCHNEID-3  19
         ERROR = .TRUE.                                                   SCHNEID-3  20
         RETURN                                                           SCHNEID-3  21
    5    CONTINUE                                                         SCHNEID-3  22
C                                                                         SCHNEID-3  23
      ERROR = .FALSE.                                                     SCHNEID-3  24
      MODEL = "POLYNOMIAL"                                                SCHNEID-3  25
C                                                                         SCHNEID-3  26
      SUMALL = 0.0                                                        SCHNEID-3  27
      SUMAVG = 0.0                                                        SCHNEID-3  28
      SUMUSE = 0.0                                                        SCHNEID-3  29
      IDOEND = NAVG - 1                                                   SCHNEID-3  30
      DO 10 I=1,IDOEND                                                    SCHNEID-3  31
         SUMAVG = SUMAVG + OBSERV(I)                                      SCHNEID-3  32
         SUMALL = SUMALL + OBSERV(I)                                      SCHNEID-3  33
   10    CONTINUE                                                         SCHNEID-3  34
      DO 12 I=NAVG,NOBS                                                   SCHNEID-3  35
         SUMUSE = SUMUSE + OBSERV(I)                                      SCHNEID-3  36
         SUMALL = SUMALL + OBSERV(I)                                      SCHNEID-3  37
   12    CONTINUE                                                         SCHNEID-3  38
      BIGA = OBSERV(NAVG)*(NAVG-1)                                        SCHNEID-3  39
      IDOEND = NOBS - NAVG                                                SCHNEID-3  40
      DO 16 I=1,IDOEND                                                    SCHNEID-3  41
         BIGA = BIGA + OBSERV(NAVG+I)*(NAVG+I-1)                          SCHNEID-3  42
   16    CONTINUE                                                         SCHNEID-3  43
C                                                                         SCHNEID-3  44
      ORDER = NOBS + NAVG                                                 SCHNEID-3  45
      NCOEF = ORDER + 1                                                   SCHNEID-3  46
      DO 18 I=1,NCOEF                                                     SCHNEID-3  47
         AT(I) = 0.0                                                      SCHNEID-3  48
   18    CONTINUE                                                         SCHNEID-3  49
      AT(NOBS+NAVG+1) = BIGA                                              SCHNEID-3  50
      AT(NOBS+NAVG) = -(BIGA + SUMUSE)                                    SCHNEID-3  51
      AT(NOBS+2) = AT(NOBS+2) - (BIGA + NAVG*SUMAVG - SUMAVG)             SEPT*77*4  16
      AT(NOBS+1) = BIGA + SUMUSE + NAVG*SUMAVG - SUMAVG                   SCHNEID-3  53
      AT(NAVG+1) = -(BIGA - NOBS*SUMALL)                                  SCHNEID-3  54
      AT(NAVG) = BIGA + SUMUSE - NOBS*SUMALL                              SCHNEID-3  55
      AT(2) = AT(2) + BIGA + NAVG*SUMAVG - NOBS*SUMALL - SUMAVG           SEPT*77*4  17
      AT(1) = -(BIGA + NAVG*SUMAVG + SUMUSE - NOBS*SUMALL - SUMAVG)       SCHNEID-3  57
C                                                                         SCHNEID-3  58
```

```
                      ALOW = 1.1                                                        SCHNEID-3  59
                      AHIGH = 2.71828                                                   SCHNEID-3  60
     60               IF (MSIGN(EVLPOLY(ALOW)).NE.MSIGN(EVLPOLY(AHIGH)))    GO TO 20   SCHNEID-3  61
                         ERROR = .TRUE.                                                 SCHNEID-3  62
                         PRINT *,"SETSCH3-- SIGN(P(",ALOW,")) = SIGN(P(",AHIGH,         SCHNEID-3  63
                     *                   ")) ",EVLPOLY(ALOW),"  ",EVLPOLY(AHIGH)      SCHNEID-3  64
                         PRINT 900,((EVLPOLY(FLOAT(I+J-11)/100.0),I=1,10),J=110,280,10) SCHNEID-3  65
     65          900     FORMAT (1H ,10F13.6)                                           SCHNEID-3  66
                         R E T U R N                                                    SCHNEID-3  67
                      20 CONTINUE                                                       SCHNEID-3  68
                      EPS = 1.0E-6                                                      SCHNEID-3  69
                      NSIG = 6                                                          SCHNEID-3  70
     70               MAXFN = 100                                                       SCHNEID-3  71
                      CALL ZBRENT (EVLPOLY,EPS,NSIG ,ALOW,AHIGH,MAXFN,IER)              SCHNEID-3  72
                      IF (IER.EQ.0)          GO TO 25                                   SCHNEID-3  73
                         ERROR = .TRUE.                                                 SCHNEID-3  74
                         PRINT *,"IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER           SCHNEID-3  75
     75                  *                  ," TO SETSCH3."                            SCHNEID-3  76
                         R E T U R N                                                    SCHNEID-3  77
                      25 CONTINUE                                                       SCHNEID-3  78
                      BETA = ALOG(AHIGH)                                                SCHNEID-3  79
                   C                                                                    SCHNEID-3  80
     80               ALPHA = (SUMALL*BETA)/(1.0-EXP(-BETA*NOBS))                       SCHNEID-3  81
                   C                                                                    SCHNEID-3  82
                      SAVE(1) = BETA                                                    SCHNEID-3  83
                      SAVE(2) = ALPHA                                                   SCHNEID-3  84
                   C                                                                    SEPT*77*4  13
     85               PRINT *,"  SCHNEIDEWIND (3) PARMS: ALPHA = ",ALPHA,               DEC*77      5
                     *        "  BETA = ",BETA                                         OCT*77*2   14
                      CALL SUMSQHE (ESTSCH,NOBS,OBSERV,SAVE,1,0.0)                      OCT*77*2   15
                      CALL SUMSQHE (ESTSCH,NOBS,OBSERV,SAVE,1,BETA)                     OCT*77*2   16
                      PRINT *,"                              WITH ",MAXFN," EVALUATIONS."  SEPT*77*4  24
     90               C                                                                SEPT*77*4  25
                      R E T U R N                                                       SCHNEID-3  85
                        E N D                                                           SCHNEID-3  86
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 SETSCH3

VARIABLES	SN	TYPE			RELOCATION				
357	AHIGH	REAL			356	ALOW	REAL		
350	ALPHA	REAL			2	AT	REAL	ARRAY	PASS
355	BETA	REAL			354	BIGA	REAL		
361	EPS	REAL			0	ERROR	LOGICAL		F.P.
353	I	INTEGER			352	IDDEND	INTEGER		
364	IER	INTEGER			360	J	INTEGER		
363	MAXFN	INTEGER			0	MODEL	INTEGER		PASS
0	NAME	INTEGER		F.P.	355	NDDEF	INTEGER		
0	NOBS	INTEGER		F.P.	362	NSIG	INTEGER		
0	OBSERV	REAL	ARRAY	F.P.	1	ORDER	INTEGER		PASS
0	SAVE	REAL	ARRAY	F.P.	347	SUMALL	REAL		

```
VARIABLES     SN  TYPE          RELOCATION
  350  SUMAVG     REAL                                351  SUMUSE     REAL

FILE NAMES          MODE
     OUTPUT         MIXED

EXTERNALS           TYPE    ARGS
     ALOG           REAL       1 LIBRARY                ESTSCH                0
     EVLPOLY        REAL       1                        EXP           REAL    1 LIBRARY
     MSIGN          INTEGER    1                        SUMSQWE               6
     ZBRENT                    7

INLINE FUNCTIONS    TYPE    ARGS
     FLOAT          REAL       1  INTRIN

STATEMENT LABELS
    14   5                                    0  10                                  0  12
     0  16                                    0  18                                164  20
   176  25                                  313  900      FMT

LOOPS  LABEL     INDEX      FROM-TO    LENGTH    PROPERTIES
    26  10       I          30 33        48      INSTACK
    36  12       I          34 37        48      INSTACK
    53  16       I          40 42        48      INSTACK
    70  18       I          46 48        28      INSTACK
   144          * J          64 64       168              EXT REFS  NOT INNER
   143          * I          64 64       138              EXT REFS

COMMON BLOCKS    LENGTH
     PASS            52

STATISTICS
  PROGRAM LENGTH                  437B     287
  CM LABELED COMMON LENGTH         64B      52
```

```
      SUBROUTINE SETJME (NOBS, OBSERV, SAVE, ERROR)                    JMEXP    4
      EXTERNAL EVLJME,ESTJME                                           DEC*77   6
      LOGICAL ERROR                                                    JMEXP    6
      REAL SAVE(2),OBSERV(NOBS)                                        JMEXP    7
      COMMON /PASS/ MODEL,BIGP,NUMB                                    JMEXP    8
C                                                                      JMEXP    9
C--->    ACCEPTS 'NOBS' OBSERVATIONS OF TIME BETWEEN SOFTWARE          JMEXP   10
C        ERRORS IN ROW 'OBSERV', COMPUTES THE PARAMETERS OF THE        JMEXP   11
C        JELINSKI-MORANDA EXPONENTIAL MODEL, AND RETURNS THE           JMEXP   12
C        PARAMETER VALUES IN 'SAVE'.  'SAVE(1)' = ESTIMATE OF PHI      JMEXP   13
C                                     'SAVE(2)' = ESTIMATE OF 'K'      JMEXP   14
C        THE MODEL IS Z(I)(T) = PHI * (N-(I-1,))                       JMEXP   15
C                                                                      JMEXP   16
C--->    THE ESTIMATOR FUNCTION THAT MUST BE SOLVED FOR 'N'=0,0 IS     JMEXP   17
C        EVALUATED BY 'EVLJME'.  ITS AUXULARY PARAMETERS ARE           JMEXP   18
C        COMPUTED AND STORED IN /PASS/.                                JMEXP   19
C                                                                      JMEXP   20
      ERROR = .FALSE.                                                  JMEXP   21
      MODEL = "JMEXP"                                                  JMEXP   22
C                                                                      JMEXP   23
      SUM1 = 0.0                                                       JMEXP   24
      SUM2 = OBSERV(1)                                                 JMEXP   25
      DO 10 I=2,NOBS                                                   JMEXP   26
         SUM1 = SUM1 + OBSERV(I)*(I-1)                                 JMEXP   27
         SUM2 = SUM2 + OBSERV(I)                                       JMEXP   28
   10    CONTINUE                                                      JMEXP   29
      BIGP = SUM1/SUM2                                                 JMEXP   30
      NUMB = NOBS                                                      JMEXP   31
C                                                                      JMEXP   32
      CALL FINDBND (EVLJME,ALOW,AHIGH,FLOAT(NOBS-1),0.5)               JMEXP   33
      EPS = 1.0E-6                                                     JMEXP   34
      NSIG = 8                                                         JMEXP   35
      MAXFN = 75                                                       JMEXP   36
      CALL ZBRENT (EVLJME,EPS,NSIG,ALOW,AHIGH,MAXFN,IER)               JMEXP   37
      IF (IER.EQ.0)          GO TO 20                                  JMEXP   38
         ERROR = .TRUE.                                                JMEXP   39
         PRINT *," IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER         JMEXP   40
     *                  ," TO SETJME."                                 JMEXP   41
         R E T U R N                                                   JMEXP   42
   20    CONTINUE                                                      JMEXP   43
C                                                                      JMEXP   44
      HATN = AHIGH                                                     JMEXP   45
      HPHI = NOBS / (HATN*SUM2 - SUM1)                                 JMEXP   46
      SAVE(1) = HPHI                                                   OCT*77*2 17
      SAVE(2) = HATN                                                   OCT*77*2 18
      PRINT *,"JMEXP PARMS: PHI = ",HPHI,"  N HAT = ",HATN             JMEXP   47
      CALL SUMSQHE (ESTJME, NOBS, OBSERV, SAVE, 2 , 0.0)               DEC*77   7
      PRINT *,"           WITH ",MAXFN," EVALUATIONS"                  JMEXP   48
      R E T U R N                                                      JMEXP   51
         E N D                                                         JMEXP   52
```

SYMBOLIC REFERENCE MAP (R=1)

```
ENTRY POINTS
     3  SETJME

VARIABLES      SN  TYPE              RELOCATION
   144  AHIGH      REAL                                143  ALOW       REAL
     1  BIGP       REAL                    PASS        145  EPS        REAL
     0  ERROR      LOGICAL                 F.P.        151  HATN       REAL
   152  HPHI       REAL                                142  I          INTEGER
   150  IER        INTEGER                             147  MAXFN      INTEGER
     0  MODEL      INTEGER                 PASS          0  NOBS       INTEGER          F.P.
   146  NSIG       INTEGER                               2  NUMB       INTEGER          PASS
     0  OBSERV     REAL          ARRAY     F.P.          0  SAVE       REAL       ARRAY F.P.
   140  SUM1       REAL                                141  SUM2       REAL

FILE NAMES      MODE
     OUTPUT     FREE

EXTERNALS          TYPE   ARGS
     ESTJME                 0                        EVLJME              0
     FINDBND                5                        SUMSQME             6
     ZBRENT                 7

INLINE FUNCTIONS   TYPE   ARGS
     FLOAT      REAL       1  INTRIN

STATEMENT LABELS
     0  10                                  45  20

LOOPS  LABEL   INDEX     FROM-TO    LENGTH     PROPERTIES
    17  10     I          23 26        58       INSTACK

COMMON BLOCKS   LENGTH
     PASS            3

STATISTICS
  PROGRAM LENGTH                  1768     126
  CM LABELED COMMON LENGTH          38       3
```

```
      1                FUNCTION EVLJME (BIGN)                                    JMEXP     53
                       COMMON /PASS/ MODEL,BIGP,NOBS                             JMEXP     54
                  C                                                              JMEXP     55
                       IF (MODEL.NE."JMEXP")   PRINT *," ***** BAD CALL TO EVLJME *****"  JMEXP     56
      5           C                                                              JMEXP     57
                       SUM = BIGP/BIGN                                           JMEXP     58
                       NOBSM1 = NOBS - 1                                         JMEXP     59
                       DO 10 I=1,NOBSM1                                          JMEXP     60
                          SUM = SUM + (BIGP-I)/(BIGN-I)                          JMEXP     61
     10                10    CONTINUE                                           JMEXP     62
                       EVLJME = SUM                                              JMEXP     63
                       R E T U R N                                               JMEXP     64
                          E N D                                                  JMEXP     65
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
4 EVLJME

VARIABLES	SN	TYPE	RELOCATION				
0	BIGN	REAL	F.P.	1	BIGP	REAL	PASS
36	EVLJME	REAL		41	I	INTEGER	
0	MODEL	INTEGER	PASS	2	NOBS	INTEGER	PASS
40	NOBSM1	INTEGER		37	SUM	REAL	

FILE NAMES	MODE
OUTPUT	FREE

STATEMENT LABELS
0 10

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
21	10	I	6 10	68	INSTACK

COMMON BLOCKS	LENGTH
PASS	3

STATISTICS

PROGRAM LENGTH	478	39
CM LABELED COMMON LENGTH	38	3

```
      1                 SUBROUTINE ESTJME (I,SAVE,LAMDA,MTTE)                    JMEXP    66
                        REAL LAMDA,MTTE,SAVE(2)                                  JMEXP    67
                  C                                                              JMEXP    68
                  C--->    ACCEPTS 'I' -- NUMBER OF ERRORS TO DATE, 'SAVE', A ROW OF   JMEXP  69
      5           C        PARAMETERS COMPUTED BY SETJME, AND APPLYS THE JELINSKI-MORANDA  JMEXP  70
                  C        EXPONENTIAL MODEL TO GIVE 'LAMDA' A ESTIMATE OF THE CURRENT   JMEXP  71
                  C        ERROR RATE, AND 'MTTE' AN ESTIMATE OF TIME TO NEXT ERROR.     JMEXP  72
                  C                                                              JMEXP    73
                        HPHI = SAVE(1)                                           JMEXP    74
     10                 HATN = SAVE(2)                                           JMEXP    75
                        LAMDA = HPHI * (HATN - (I-1))                            JMEXP    76
                        MTTE = 1.0/LAMDA                                         JMEXP    77
                        R E T U R N                                              JMEXP    78
                           E N D                                                 JMEXP    79
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS

3 ESTJME

VARIABLES	SN	TYPE	RELOCATION						
20	HATN	REAL		17	HPHI	REAL			
0	I	INTEGER	F.P.	0	LAMDA	REAL		F.P.	
0	MTTE	REAL	F.P.	0	SAVE	REAL	ARRAY	F.P.	

STATISTICS

PROGRAM LENGTH 218 17

```
      SUBROUTINE SETEXTE (NOBS, OBSERV,SAVE,ERROR)                      EXTJMEXP  2
      EXTERNAL EVLEXTE,ESTEXTE                                          SEPT*77*4 26
      LOGICAL ERROR                                                     EXTJMEXP  4
      REAL SAVE(2),OBSERV(NOBS)                                         SEPT*77*3 26
      COMMON /PASS/ MODEL,BIGR,NUMB,VALUE(50)                           SEPT*77*3 27
C                                                                       EXTJMEXP  7
C---->    ACCEPTS 'NOBS' OBSERVATIONS OF ERRORS PER INTERVAL IN         EXTJMEXP  8
C         ROW 'OBSERV', COMPUTES THE PARAMETERS OF THE JELINSKI-        EXTJMEXP  9
C         MORANDA EXPONENTIAL MODEL AS EXTENDED BY LIPOV, AND           EXTJMEXP 10
C         RETURNS THE PARAMETER VALUES IN 'SAVE',                       EXTJMEXP 11
C                                        'SAVE(1)' = ESTIMATE OF PHI    EXTJMEXP 12
C                                        'SAVE(2)' = ESTIMATE OF 'N'    EXTJMEXP 13
C         THE MODEL IS Z(I)(T) = PHI * (N-LITTLE N(I-1))                EXTJMEXP 14
C                                                                       EXTJMEXP 15
C---->    THE ESTIMATOR FUNCTION THAT MUST BE SOLVED FOR 'N'=0.0 IS     EXTJMEXP 16
C         EVALUATED BY 'EVLEXTE'. ITS AUXILARY PARAMETERS ARE           EXTJMEXP 17
C         COMPUTED AND STORED IN /PASS/.                                EXTJMEXP 18
C                                                                       EXTJMEXP 21
      ERROR = .FALSE.                                                   EXTJMEXP 22
      MODEL = "EXTJMEXP"                                                EXTJMEXP 23
      TIME = 1.0                                                        EXTJMEXP 24
      NUMB = NOBS                                                       SEPT*77*3 28
C                                                                       EXTJMEXP 25
      CUMERS = 0.0                                                      EXTJMEXP 26
      SUM1 = 0.0                                                        EXTJMEXP 27
      SUM2 = 1.0                                                        EXTJMEXP 28
      VALUE(1) = OBSERV(1)                                              SEPT*77*3 29
      DO 10 I=2,NOBS                                                    EXTJMEXP 29
         VALUE(I) = OBSERV(I)                                           SEPT*77*3 30
         CUMERS = CUMERS + OBSERV(I-1)                                  EXTJMEXP 30
         SUM1 = SUM1 + CUMERS*TIME                                      EXTJMEXP 31
         SUM2 = SUM2 + TIME                                             EXTJMEXP 32
   10    CONTINUE                                                       EXTJMEXP 33
      BIGR = SUM1/SUM2                                                  EXTJMEXP 34
C                                                                       EXTJMEXP 35
      CALL FINOBND(EVLEXTE,ALOW,AHIGH,CUMERS,10.0)                      EXTJMEXP 36
      EPS = 1.0E-6                                                      EXTJMEXP 37
      NSIG = 6                                                          EXTJMEXP 38
      MAXFN = 75                                                        EXTJMEXP 39
      CALL ZBRENT (EVLEXTE,EPS,NSIG,ALOW,AHIGH,MAXFN,IER)               EXTJMEXP 40
      IF (IER.EQ.0)           GO TO 20                                  EXTJMEXP 41
         ERROR = .TRUE.                                                 EXTJMEXP 42
         PRINT *," IMSL ROUTINE ZBRENT RETURNS ERROR NO. ",IER          EXTJMEXP 43
     *                  ," TO SETEXTE."                                EXTJMEXP 44
         R E T U R N                                                    EXTJMEXP 45
   20    CONTINUE                                                       EXTJMEXP 46
C                                                                       EXTJMEXP 47
      MAIN = AHIGH                                                      EXTJMEXP 48
      SUM = MAIN*TIME                                                   EXTJMEXP 49
      CUMERS = 0.0                                                      EXTJMEXP 50
      DO 30 I=2,NOBS                                                    EXTJMEXP 51
         CUMERS = CUMERS + OBSERV(I-1)                                  EXTJMEXP 52
         SUM = SUM + (MAIN-CUMERS)*TIME                                 EXTJMEXP 53
   30    CONTINUE                                                       EXTJMEXP 54
      MPHI = (CUMERS+OBSERV(NOBS))/SUM                                  EXTJMEXP 55
C                                                                       OCT*77*1  7
      SAVE(1) = MPHI                                                    OCT*77*1  8
```

```
                      SAVE(2) = HATN                                                    OCT*77*1    9
                C                                                                       SEPT*77*4  27
     60         C                                                                      EXTJMEXP   56
                      PRINT *," EXTEXP PARMS! PHI = ",HPHI," N HAT = ",HATN             EXTJMEXP   57
                      CALL SUMSQWE (ESTEXTE, NOBS, OBSERV, SAVE, 21 , 0.0)              DEC*77      8
                      CALL SUMSQWE (ESTEXTE, NOBS, OBSERV, SAVE, 11 , 0.0)              DEC*77      9
                      PRINT *,"                WITH ",MAXFN," EVALUATIONS"              EXTJMEXP   56
     65               R E T U R N                                                       EXTJMEXP   61
                        E N D                                                           EXTJMEXP   62
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 SETEXTE

VARIABLES	SN	TYPE		RELOCATION					
205	AMIGH	REAL			204	ALOW	REAL		
1	BIGR	REAL		PASS	200	CUMERS	REAL		
206	EPS	REAL			0	ERROR	LOGICAL		F.P.
212	HATN	REAL			214	HPHI	REAL		
203	I	INTEGER			211	IER	INTEGER		
210	MAXFN	INTEGER			0	MODEL	INTEGER		PASS
0	NOBS	INTEGER		F.P.	207	NSIG	INTEGER		
2	NUMB	INTEGER		PASS	0	OBSERV	REAL	ARRAY	F.P.
0	SAVE	REAL	ARRAY	F.P.	213	SUM	REAL		
201	SUM1	REAL			202	SUM2	REAL		
177	TIME	REAL			3	VALUE	REAL	ARRAY	PASS

FILE NAMES	MODE
OUTPUT	FREE

EXTERNALS	TYPE	ARGS			
ESTEXTE		0		EVLEXTE	0
FINDGND		5		SUMSQWE	6
ZBRENT		7			

STATEMENT LABELS
0 10 50 20 0 30

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
22	10	I	28 33	108	OPT
60	30	I	51 54	68	INSTACK

COMMON BLOCKS	LENGTH
PASS	53

STATISTICS

PROGRAM LENGTH	2418	161
CM LABELED COMMON LENGTH	658	53

```
      1                FUNCTION EVLEXTE (BIGN)                                  EXTJMEXP   63
                       COMMON /PASS/ MODEL,BIGR,NOBS,OBSER/(50)                 SEPT*77*3  31
                 C                                                              EXTJMEXP   66
                       IF (MODEL.NE."EXTJMEXP")  PRINT *,"***** BAD CALL TO EVLEXTE ***" EXTJMEXP   67
      5          C                                                              EXTJMEXP   68
                       SUM = OBSERV(1)/BIGN                                     EXTJMEXP   69
                       CUMERS = 0.0                                             EXTJMEXP   70
                       DO 10 I=2,NOBS                                           EXTJMEXP   71
                          CUMERS = CUMERS + OBSERV(I-1)                         EXTJMEXP   72
     10                   SUM = SUM + OBSER/(I)/(BIGN-CUMERS)                   EXTJMEXP   73
                    10    CONTINUE                                             EXTJMEXP   74
                       EVLEXTE = SUM*(BIGN-BIGR) - (CUMERS+OBSERV(NOBS))        EXTJMEXP   75
                 C                                                              EXTJMEXP   76
                       R E T U R N                                              EXTJMEXP   77
     15                E N D                                                    EXTJMEXP   78
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
4 EVLEXTE

VARIABLES	SN	TYPE	RELOCATION					
0	BIGN	REAL	F.P.	1	BIGR	REAL		PASS
43	CUMERS	REAL		41	EVLEXTE	REAL		
44	I	INTEGER		0	MODEL	INTEGER		PASS
2	NOBS	INTEGER	PASS	3	OBSERV	REAL	ARRAY	PASS
42	SUM	REAL						

FILE NAMES	MODE
OUTPUT	FREE

STATEMENT LABELS
0 10

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES
20	10	I	8 11	68	INSTACK

COMMON BLOCKS	LENGTH
PASS	53

STATISTICS

PROGRAM LENGTH	518	41
CM LABELED COMMON LENGTH	658	53

```
      1              SUBROUTINE ESTEXTE (LITTLEN,SAVE,ERRORS,MTBE)                EXTJMEXP   79
                     REAL LITTLEN,SAVE(2),MTBE                                    EXTJMEXP   80
                C                                                                 EXTJMEXP   81
                C----->   ACCEPTS 'LITTLEN' -- NUMBER OF ERRORS TO DATE, 'SAVE',  EXTJMEXP   82
      5         C         A ROW OF PARAMETERS COMPUTED BY SETEXTE, AND APPLYS THE  EXTJMEXP   83
                C         EXTENDED JELINSKI-MORANDA EXPONENTIAL MODEL TO GIVE     EXTJMEXP   84
                C         'ERRORS' A ESTIMATE OF ERRORS TO BE FOUND IN THE        EXTJMEXP   85
                C         NEXT DEBUGGING INTERVAL, AND 'MTBE' AN ESTIMATE OF THE  EXTJMEXP   86
                C         MEAN TIME BETWEEN ERRORS DURING THAT INTERVAL.          EXTJMEXP   87
     10         C                                                                 EXTJMEXP   88
                     MPHI = SAVE(1)                                               EXTJMEXP   89
                     MAIN = SAVE(2)                                               EXTJMEXP   90
                     ERRORS = MPHI * (MAIN - LITTLEN)                             EXTJMEXP   91
                     MTBE = 1.0/ERRORS                                            EXTJMEXP   92
     15              R E T U R N                                                  EXTJMEXP   93
                        E N D                                                     EXTJMEXP   94
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 ESTEXTE

VARIABLES	SN	TYPE	RELOCATION						
0	ERRORS	REAL	F.P.	16	MAIN	REAL			
15	MPHI	REAL		0	LITTLEN	REAL		F.P.	
0	MTBE	REAL	F.P.	0	SAVE	REAL	ARRAY	F.P.	

STATISTICS
PROGRAM LENGTH 178 15

```
      SUBROUTINE FINDBND (F,ALOW,AHIGH,BASE,FACT)                       UTILITY    2
C                                                                       UTILITY    3
C--->   FINDBND TRYS TO LOCATE THE FIRST SIGN CHANGE IN THE             UTILITY    4
C       FUNCTION F(X) ABOVE THE VALUE 'BASE'.                           UTILITY    5
C       'AHIGH' AND 'ALOW' ARE TO BE POSITIONED ON EITHER SIDE          UTILITY    6
C       OF IT.  THE UPPER BOUND IS RAISED BY I*'FACT' EACH TEST.        UTILITY    7
C                                                                       UTILITY    8
      ALOW = BASE + .001                                                UTILITY    9
      AHIGH = BASE + 1.0                                                UTILITY   10
      DO 10 I=1,100                                                     UTILITY   11
         IF (MSIGN(F(ALOW)).NE.MSIGN(F(AHIGH)))  GO TO 20               UTILITY   12
         AHIGH = AHIGH + FACT*I                                         UTILITY   13
   10 CONTINUE                                                          UTILITY   14
C                                                                       UTILITY   15
      PRINT *," FINDBND CAN NOT FIND ZERO CROSSING"                     UTILITY   16
      PRINT *,ALOW,AHIGH,BASE,FACT,F(ALOW),F(AHIGH)                     UTILITY   17
   20 CONTINUE                                                          UTILITY   18
      R E T U R N                                                       UTILITY   19
        E N D                                                           UTILITY   20
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 FINDBND

VARIABLES	SN	TYPE	RELOCATION				
0	AHIGH	REAL	F.P.	0	ALOW	REAL	F.P.
0	BASE	REAL	F.P.	0	FACT	REAL	F.P.
110	I	INTEGER					

FILE NAMES	MODE
OUTPUT	FREE

EXTERNALS	TYPE	ARGS			
F	REAL	1	F.P.	MSIGN	INTEGER 1

STATEMENT LABELS
0 10 56 20

LOOPS	LABEL	INDEX	FROM-TO	LENGTH	PROPERTIES	
23	10	* I	10 13	218	EXT REFS	EXITS

STATISTICS
PROGRAM LENGTH 1348 92

```
      1                 FUNCTION MSIGN (X)                                    UTILITY    21
                        C                                                     UTILITY    22
                        C--->   RETURNS A INTEGER WITH THE VALUE '-1', '0', OR '1'  UTILITY    23
                        C       DEPENDING ON THE SIGN OF ITS PARAMETER 'X'.  THIS ALLOWS  UTILITY    24
      5                 C       TESTS FOR SAME OR DIFFERENT SIGNS.            UTILITY    25
                        C                                                     UTILITY    26
                              MSIGN = 0                                       UTILITY    27
                              IF (X.LT.0.0)     MSIGN = -1                    UTILITY    28
                              IF (X.GT.0.0)     MSIGN =  1                    UTILITY    29
     10                       RETURN                                          UTILITY    30
                                 END                                          UTILITY    31
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
  + MSIGN

VARIABLES	SN	TYPE	RELOCATION				
15 MSIGN		INTEGER		0	X	REAL	F.P.

STATISTICS
  PROGRAM LENGTH 168 14

```
      SUBROUTINE SUMSQRE (F, NOBS, OBSERV, SAVE, IUSE, WEIGHT)          OCT*77*2  20
      INTEGER  ISPARM(3)                                                OCT*77*1  10
      REAL OBSERV(NOBS),SAVE(2),RESULT(2),ASPARM(3)                     OCT*77*1  11
      EQUIVALENCE (ASPARM(1),ISPARM(1))                                 OCT*77*1  12
      SQUARE(X) = X*X                                                   UTILITY   34
C                                                                       UTILITY   35
C---->   INPUTS ARE ACTUAL OBSERVATIONS IN 'OBSERV' AND A               UTILITY   36
C        SUBROUTINE 'F' WHICH GIVES A MODEL'S PREDICTIONS.              UTILITY   37
C        'IUSE' INDICATES THE TYPE OF OBSERVATIONS IN 'OBSERV'          OCT*77*1  13
C                  X1 = LAMDA (ERRORS)                                  OCT*77*1  14
C                  X2 = MTTE                                            OCT*77*1  15
C        'IUSE' ALSO INDICATES WHAT VALUES TO PASS TO 'F'               OCT*77*1  16
C        AS ITS FIRST PARAMETER    ('ISPAR')                            OCT*77*1  17
C                  0X = I (THE INTERVAL NUMBER)                         OCT*77*1  18
C                  1X = ACTUAL ERRORS TO DATE                           OCT*77*1  19
C                  2X = MODEL PREDICTED ERRORS TO DATE                  OCT*77*1  20
C        'WEIGHT' IS THE PARAMETER OF A WEIGHTING FUNCTION              UTILITY   38
C                  WT = EXP(WEIGHT*TIME TO DATE)                        UTILITY   39
C        IF 'WEIGHT' IS ZERO THE RESULT IS A SIMPLE SUM OF              UTILITY   40
C        SQUARES.  THE RESULT IS RETURNED IN 'SUMSQ'.                   UTILITY   41
C                                                                       UTILITY   42
      WT = 1.0                                                          UTILITY   43
      SUM = 0.0                                                         UTILITY   44
      ASPARM(2) = 0.0                                                   OCT*77*1  21
      ASPARM(3) = 0.0                                                   OCT*77*1  22
      ISPAR = IUSE/10                                                   OCT*77*1  23
      ISOBS = IUSE - (ISPAR*10)                                         OCT*77*1  24
      DO 10 I=1,NOBS                                                    UTILITY   45
         IF (WEIGHT.NE.0.0)                    WT = EXP(WEIGHT*I)       UTILITY   46
         ISPARM(1) = I                                                  OCT*77*1  25
         CALL F(ISPARM(ISPAR+1),SAVE,RESULT(1),RESULT(2))               OCT*77*1  26
C              ****** NOTE: IF ISPAR = 0 INTEGER; IF =1 OR 2 IT'S REAL  OCT*77*1  27
         SUM = SUM + WT*SQUARE(OBSERV(I) - RESULT(ISOBS))/RESULT(ISOBS) OCT*77*1  28
         ASPARM(2) = ASPARM(2) + OBSERV(I)                              OCT*77*1  29
         ASPARM(3) = ASPARM(3) + RESULT(1)                              OCT*77*1  30
   10    CONTINUE                                                       UTILITY   49
C                                                                       UTILITY   50
      IF (ISPAR.NE.1)                      GO TO 30                     OCT*77*2  21
         PRINT 900,SUM                                                  OCT*77*2  22
         RETURN                                                         OCT*77*2  23
C                                                                       OCT*77*2  24
   30 IF (WEIGHT.EQ.0.0)                   GO TO 40                     OCT*77*2  25
         PRINT 910,SUM                                                  OCT*77*2  26
         RETURN                                                         OCT*77*2  27
C                                                                       OCT*77*2  28
   40 PRINT 920,SUM                                                     OCT*77*2  29
      RETURN                                                            UTILITY   52
  900 FORMAT (1H+,95X," ACT = ",F10.6)                                  OCT*77*2  30
  910 FORMAT (1H+,95X," WGT = ",F10.6)                                  OCT*77*2  31
  920 FORMAT (1H+,75X,"CHSQ = ",F10.6)                                  OCT*77*2  32
      END                                                               UTILITY   53
```

```
        SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
     3  SUMSQWE

VARIABLES     SN  TYPE            RELOCATION
   137  ASPARM     REAL      ARRAY                  136  I          INTEGER
   135  ISOBS      INTEGER                          134  ISPAR      INTEGER
   137  ISPARM     INTEGER   ARRAY                    0  IUSE       INTEGER                F.P.
     0  NOBS       INTEGER             F.P.           0  OBSERV     REAL       ARRAY       F.P.
   142  RESULT     REAL      ARRAY                    0  SAVE       REAL       ARRAY       F.P.
   133  SUM        REAL                               0  WEIGHT     REAL                   F.P.
   132  WT         REAL

FILE NAMES        MODE
     OUTPUT       FMT

EXTERNALS         TYPE   ARGS
     EXP          REAL     1  LIBRARY                  F                  4  F.P.

INLINE FUNCTIONS  TYPE   ARGS
     SQUARE       REAL     1    SF

STATEMENT LABELS
     0  10                                  61  30                                    65  40
   113  900       FMT                      121  910       FMT                        125  920       FMT

LOOPS  LABEL     INDEX      FROM-TO     LENGTH     PROPERTIES
    2o  10       * I         28  36        278                EXT REFS

STATISTICS
  PROGRAM LENGTH                1608     112
```

```
      1                 SUBROUTINE FINDCRS (F, ISIGN, FLEX, START, STEPFAC)                    OCT*77*1   31
                      C                                                                        OCT*77*1   32
                      C----->   LOCATES A CHANGE OF SIGN KNOWN TO BE IN THE VICINITY           OCT*77*1   33
                      C         OF 'START'.  THIS PROCEDURE IS USED TO LOCATE THAT             OCT*77*1   34
      5               C         ROOT AND SEPERATE IT FROM ANOTHER KNOWN TO EXIST AT            OCT*77*1   35
                      C         'START' + SUM(STEP/(10*I))  THE VALUES TRIED FORM A            OCT*77*1   36
                      C         SERIES.     START     STEP          --  VALUES  --             OCT*77*1   37
                      C                       .9        .9       .9    .99   .999  .9999 .99999 OCT*77*1   38
                      C                      1.1       -.9      1.1   1.01  1.001 1.0001   . . . OCT*77*1   39
     10               C                                                                        OCT*77*1   40
                            STEP = STEPFAC                                                     OCT*77*1   41
                            FLEX = START                                                       OCT*77*1   42
                            DO 10 I=1,13                                                       OCT*77*1   43
                               IF (MSIGN(F(FLEX)).NE.ISIGN)              GO TO 20              OCT*77*1   44
     15                        STEP = STEP/10.0                                              OCT*77*1   45
                               FLEX = FLEX + STEP                                            OCT*77*1   46
                         10    CONTINUE                                                      OCT*77*1   47
                      C                                                                        OCT*77*1   48
                            PRINT *,"FINDCRS CANNOT FIND SIGN CHANGE"                          OCT*77*1   49
     20                     PRINT *,FLEX,F(FLEX),ISIGN                                        OCT*77*1   50
                            STEP = STEPFAC                                                     OCT*77*1   51
                            FLEX = START                                                       OCT*77*1   52
                            DO 15 I=1,13                                                       OCT*77*1   53
                               PRINT *,FLEX,"      ",F(FLEX)                                   OCT*77*1   54
     25                        STEP = STEP/10.                                               OCT*77*1   55
                               FLEX = FLEX + STEP                                            OCT*77*1   56
                         15    CONTINUE                                                      OCT*77*1   57
                      C                                                                        OCT*77*1   58
                         20 CONTINUE                                                         OCT*77*1   59
     30                     RETURN                                                           OCT*77*1   60
                            END                                                              OCT*77*1   61
```

SYMBOLIC REFERENCE MAP (R=1)

ENTRY POINTS
3 FINDCRS

VARIABLES	SN	TYPE	RELOCATION					
0	FLEX	REAL	F.P.	112	I	INTEGER		
0	ISIGN	INTEGER	F.P.	0	START	REAL	F.P.	
111	STEP	REAL		0	STEPFAC	REAL	F.P.	

FILE NAMES	MODE
OUTPUT	FREE

EXTERNALS	TYPE	ARGS					
F	REAL	1	F.P.	MSIGN	INTEGER	1	

STATEMENT LABELS
0 10 0 15 63 20

LOOPS	LABEL		INDEX	FROM-TO	LENGTH	PROPERTIES	
22	10	*	I	13 17	148	EXT REFS	EXITS
51	15	*	I	23 27	128	EXT REFS	

STATISTICS

PROGRAM LENGTH	1418	97

FWA OF THE LOAD 101
LWA+1 OF THE LOAD 45326

WRITTEN TO FILE SOFTOBJ

TRANSFER ADDRESS -- SOFTW 6534

PROGRAM AND BLOCK ASSIGNMENTS.

BLOCK	ADDRESS	LENGTH	FILE	DATE	PROCSSR	VER LEVEL	HARDWARE	COMMENTS
/PASS/	101	256						
SOFTW	357	22761	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETJMG	23340	250	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETJMGP	23610	246	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
EVLPOLY	24056	45	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
ESTGEOM	24123	27	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETSCH1	24152	262	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
ESTSCH	24434	42	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETSCH2	24476	414	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETSCH3	25112	437	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETJME	25551	176	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
EVLJME	25747	47	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
ESTJME	26016	21	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SETEXTE	26037	241	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
EVLEXTE	26300	51	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
ESTEXTE	26351	17	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
FINDBND	26370	134	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
MSIGN	26524	16	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
SUMSQRE	26542	160	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
FINDCRS	26722	141	LGO	12/15/77	FTN	4.5 414	636X I	OPT=1 TRACE
UERTST	27053	72	UL-IMSL	07/26/77	FTN	4.5 414	636X I	OPT=1
ZBRENT	27155	263	UL-IMSL	07/26/77	FTN	4.5 414	636X I	OPT=1
/FCL.C./	27440	23						
/O3.IO./	27463	131						
GBNTRY=	27614	0	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		FCL INITILALIZATION ROUTINE.
FSCMSK=	27614	41	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		INITIALIZE CONSTANTS.
FLTOUT=	27655	310	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		COMMON FLOATING OUTPUT CODE
FORSYS=	30165	601	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		FORTRAN OBJECT LIBRARY UTILITIES.
INCOM=	30766	277	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		COMMON INPUT FORMATTING CODE
INPC=	31205	160	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		FORMATTED READ FORTRAN RECORD.
KRAKER=	31445	406	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		PROCESS FORMATTED FORTRAN INPUT.
LDIN=	32053	260	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		LIST DIRECTED INPUT FORMATTING
OUTC=	32333	172	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		FORMATTED WRITE FORTRAN RECORD.
OUTF=	32525	161	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		LIST DIRECTED OUTPUT CONTROL
GOTOER=	32706	14	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		COMPUTED GO TO ERROR PROCESSOR.
ALOG	32722	73	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		COMPUTE COMMON AND NATURAL LOGARITHMS. OPT=ALL
EXP	33015	75	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		EXPONENTIAL FUNCTION. E TO POWER X. OPT=ALL.
SYSAID=	33112	1	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		LINK BETWEEN SYS=AID AND INITIALIZATION CODE.
XTOI=	33113	10	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		REAL TO INTEGER EXPONENTIATION.
BACKSP=	33123	50	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		BACKSPACE LOGICAL RECORD.
COMIO=	33201	04	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		COMMON CODED I/O ROUTINES AND CONSTANTS.
EOF	33205	16	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		TEST FOR END OF FILE STATUS.
FLTIN=	33303	154	SL-FORTRAN	04/18/77	COMPASS	3. 2-414		COMMON FLOATING INPUT CONVERTER.

FMTAP=	33457	352	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	CRACK APLIST AND FORMAT FOR KODER/KRAKER.
FORUTL=	34031	16	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	FCL MISC. UTILITIES.
GETFIT=	34047	42	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	LOCATE AN FIT GIVEN A FILE NAME.
INPF=	34111	200	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	LIST DIRECTED INPUT CONTROL
KODER=	34311	456	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	OUTPUT FORMAT INTERPRETER.
LDOUT=	34767	241	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	LIST DIRECTED OUTPUT FORMATTING
/IO.BUF./	35230	227				
OUTB=	35457	203	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	BINARY WRITE FORTRAN RECORD.
OUTCOM=	35662	154	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	COMMON OUTPUT CODE
SYS=1ST	36036	62	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	MATH LIBRARY LINK TO ERROR MESSAGE PROCESSOR.
XTOI*	36120	26	SL-FORTRAN	04/18/77	COMPASS 3. 2-414	REAL BASE TO INTEGER POWER.
/CON.RM/	36146	6				
CIO.RM	36154	40	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/ADB.RM/	36214	10				
MOVE.RM	36224	64	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
MCT.RM	36310	227	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/JMPS.RM/	36537	11				
/MEMC.RM/	36550	3				
/OPES.FO/	36553	1				
/OPEN.FO/	36554	7				
OPEN.RM	36563	235	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/CLSF.FO/	37020	7				
CLSF.SQ	37027	131	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/CLSV.FO/	37160	7				
CLSV.SQ	37167	123	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/REW.FO/	37312	7				
REW.SQ	37321	31	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/TERM.RM/	37352	1				
/GET.FO/	37353	7				
/GET.BT/	37362	5				
/GET.RT/	37367	11				
GET.SQ	37400	1033	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
Z.SQ	40433	101	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
FSU.SQ	40534	106	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/SKBL.FO/	40642	7				
/SKSB.FO/	40651	1				
SKSB.SQ	40652	77	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
SYS.RM	40721	37	SL-SYSIO	04/18/77	COMPASS 3. 2-414	PROCESS SYSTEM REQUEST.
/PUT.FO/	41010	7				
PUT.SQ	41017	1364	SL-SYSIO	06/08/77	COMPASS 3. 2-414	
ERR.RM	42403	404	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
CHAR.SQ	43007	7	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
OSUB.RM	43016	65	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
OPEN.SQ	43103	262	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
CPEX.SQ	43365	14	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/PUT.RT/	43401	11				
RLEQ.RM	43412	42	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
WAR.SQ	43454	260	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
CLSF.RM	43734	23	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
BTRT.SQ	43757	114	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
WEOX.SQ	44073	142	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
/SKFL.FO/	44235	7				
SKFL.SQ	44244	47	SL-SYSIO	04/15/77	COMPASS 3. 2-414	
SKBL.SQ	44313	1013	SL-SYSIO	04/15/77	COMPASS 3. 2-414	

REFERENCES

1) Akiyama, F., "An Example of Software System Debugging", Proceedings of IFIP Congress 1971, North Holland Publishing Company, 1972.

2) Alberts, D.S., "The Economics of Software Quality Assurance", National Computer Conference, 1976.

3) Amster, S.J., Davis, E.J., Dickman, B.N., and Kuoni, J.P., "An Experiment in Automatic Quality Evaluation of Software", Proceedings of the Symposium on Computer Software Engineering, October 13-15, 1976.

4) Anderson, T. and Kerr, R., "Recovery Blocks in Action: A System Supporting High Reliability", Second International Conference on Software Engineering, October 13-15, 1976.

5) Anthony, A.L. and Watson, H.K., "Techniques for Developing Analytic Models", IBM Systems Journal, No. 4, 1972.

6) Avizienis, A., "Fault-Tolerance and Fault-Intolerance: Complementary Approaches to Reliable Computing", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

7) Avizienis, A., "Approaches to Computer Reliability - Then and Now", National Computer Conference, 1976.

8) Baker, W.F., Software Data Collection and Analysis: A Real-Time System Project History, RADC-TR-77-192, June 1977.

9) Belady, L.A. and Lehman, M.M., "A Model of Large Program Development", IBM Systems Journal, No. 3, 1976.

10) Birman, A. and Joyner, W.H., "A Problem-Reduction Approach to Proving Simulation Between Programs", IEEE Transactions on Software Engineering, June 1976.

11) Bloom, S., McPheters, M.J., and Tsiang, S.H., "Software Quality Control", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

12) Boehm, B.W., McClean, R.K., and Urfrig, D.B., "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software", IEEE Transactions on Software Engineering, June 1975.

13) Boehm, B.W., "The High Cost of Software", in Practical Strategies for Developing Large Software Systems, E. Horowitz, Ed., Addison-Wesley, 1975.

14) Boehm, B.W., Brown, J.R., and Lipow, M., "Quantitative Evaluation of Software Quality", Second International Conference on Software Engineering", October 13-15, 1976

15) Boehm, B.W., "Software and its Impact: A Quantitative Assessment", Datamation, May 1973.

16) Boyer, R.S., Elspar, B., and Levitt, K.N., "SELECT -- A Formal System for Testing and Debugging Programs by Symbolic Execution", Computer Science Group, Stanford Research Institute, Menlo Park, California, 1975.

17) Brown, J.R. and Lipow, M., "Testing for Software Reliability", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

18) Buckley, F.J., "Software Testing - A report From the Field", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

19) Caplain, M., "Finding Invariant Assertions for Proving Programs", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

20) Chandy, K.M., "Bayesian Models of Design Based on Intuition", Second International Conference on Software Engineering, October 13-15, 1976.

21) Corcoran, W.J., Weingarten, H. and Zehna, P.W., "Estimating Reliability After Corrective Action", Management Science, Vol. 10, No. 4, July 1964.

22) Coutinho, J. de S. "Software Reliability Growth", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

23) Coutinho, J.de S. "Special Army Requirements for Tactical Data Systems", Proceedings of the 1976 Annual Reliability and Maintainability Symposium.

24) Craig, G.R., Hetrick, W.L. and Lipow, M., Software Reliability Study, RADC-TR-74-250, October 1974.

25) Culpepper, L.M., "A System for Reliable Engineering Software", IEEE Transactions on Software Engineering, June 1975.

26) Daly, E.B., "Management of Software Development", IEEE Transactions on Software Engineering, May 1977.

27) Damman, L., Jennington, R., Kirsten, P., Grabe, R. and Long, P., Flight Test Development and Evaluation of a Multi-mode Digital Flight Control System Implemented in an A-7D (Digitac), AFFTC-TR-76-15, June 1976.

28) D'Angelis, D. and Lauro, J.A., "Software Recovery in the Fault-Tolerant Spaceborne Computer", 1976 International Symposium on Fault-Tolerant Computing, June 21-23, 1976.

29) De Sousa, P.T. and Mathur, F.P., "Modular Redundancy Without Voters Decreases Complexity of Restoring Organ", National Computer Conference, 1977.

30) Dickson, J.C., Hesse, J.L., Kientz, A.C. and Shooman, M.L., "Quantitative Analysis of Software Reliability", Proceedings of the 1972 Annual Reliability and Maintainability Symposium.

31) Dijkstra, E.W., "Correctness Concerns and, Among Other Things, Why They are Resented", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

32) Ellingson, O.E., "Computer Program and Change Control", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

33) Elspas, B., Green, M.W. and Levitt, K.N., "Software Reliability", Computer, January/February 1971.

34) Endres, A., "An Analysis of Errors and Their Causes in System Programs", IEEE Transactions on Software Engineering, June 1975.

35) Estep, J.G., "A Software Availability and Reliability Model", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

36) Fagan, M.E., "Design and Code Inspections to Reduce Errors in Program Development" IBM Systems Journal, No. 3, 1976.

37) Forman, E.H. and Singpurwalla, N.D., "An Empirical Stopping Rule for Debugging and Testing Computer Software", Journal of the American Statistical Association, Vol. 72, No. 360, December 1977.

38) Forman, E.H., "Statistical Models and Methods for Measuring Software Reliability", Ph.D. Dissertation, Department of Statistics, George Washington University, 1975.

39) Fosdick, L.D. and Osterweil, L.J., "Data Flow Analysis in Software Reliability", Computing Surveys, Vol. 8, No. 3, September 1976.

40) Fries, M.J., Software Error Data Acquisition, RADC-TR-77-130, April 1977.

41) Funami, Y. and Halstead, M.H., "A Software Physics Analysis of Akiyama's Debugging Data", Proceedings of the Symposium on Computer Software Engineering, 1976.

42) Gammill, R.C., "Software Reliability: Philosophical Underpinnings" RAND Corporation, Santa Monica, California, October 1974.

43) Gannon, J.D. and Horning, J.J., "Language Design for Programming Reliability", IEEE Transactions on Software Engineering, June 1975.

44) Gilb, T., "Parallel Programming", Datamation, October 1974.

45) Goodenough, J.B. and Zara, R.V., "The Effect of Software Structure on Software Reliability, Modifiability, and Reusability: A Case Study and Analysis", Softech, Inc., Waltham, Massachusetts, July 1974.

46) Green, T.F., Schneidewind, N.F., Howard, G.T., and Pariseau, R.J., "Program Structures, Complexity and Error Characteristics", Proceedings of the Symposium on Computer Software Engineering, 1976.

47) Green, T.F., "Software Error Detection Model", Thesis, Naval Postgraduate School, Monterey, California, June 1975.

48) Gries, D., "An Illustration of Current Ideas on the Derivation of Correctness Proofs and Correct Programs", IEEE Transactions on Software Engineering, December 1976.

49) Halstead, M.H., Software Physics: Basic Principles, IBM Research Report RJ 1583, May 1975.

50) Hantler, S.L. and King, J.C., "An Introduction to Proving the Correctness of Programs", Computing Surveys, Vol. 8, No. 3, September 1976.

51) Hecht, H., "Fault-Tolerant Software: Motivation and Capabilities", Proceedings of the Symposium on Computer Software Engineering, 1976.

52) Hecht, H., "Can Software Benefit from Hardware Experience?", Proceedings of the 1975 Annual Reliability and Maintainability Symposium.

53) Henderson, P., "Finite State Modelling in Program Development", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

54) Hoare, C.A.R., "An Axiomatic Basis for Computer Programming", Communications of the ACM, Vol. 12, No. 10, October 1969.

55) Jelinski, Z. and Moranda, P.B., "Applications of a Probability-Based Model to a Code Reading Experiment", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

56) Jelinski, Z. and Moranda, P.B., "Software Reliability Research", 1972 International Symposium on Fault-Tolerant Computing, June 1972.

57) Johnson, J.P., Software Reliability Measurement Study, The Aerospace Corporation, El Segundo, California, Report No. TR-0076(6519)-1, December 1975.

58) Katz, S. and Manna, Z., "Towards Automatic Debugging of Programs", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

59) King, J.C., "Proving Programs to be Correct", IEEE Transactions on Computers, November 1971.

60) Kuehn, R.E., "Computer Redundancy: Design, Performance, and Future", IEEE Transactions on Reliability, February 1969.

61) LaPadula, L.J., Engineering of Quality Software Systems (Software Reliability Modeling and Measurement Techniques), RADC-TR-74-325, Vol. VIII, January 1975.

62) Laprie, J.C., "Reliability and Availability of Repairable Structures", 1975 International Symposium on Fault-Tolerant Computing, June 18-20, 1975.

63) Levitt, K.N., "The Application of Program-Proving Techniques to the Verification of Synchronization Processes", Fall Joint Computer Conference, 1972.

64) Liebergot, H., "Fault Absorption by Hardware and Software", 1975 International Symposium on Fault-Tolerant Computing, June 18-20, 1975.

65) Lipow, M. and Thayer, T.A., "Prediction of Software Failures", Proceedings of the 1977 Annual Reliability and Maintainability Symposium.

66) Liskov, "A Design Methodology for Reliable Software Systems", Fall Joint Computer Conference, 1972.

67) Littlewood, B. and Verrall, J.L., "A Bayesian Reliability Growth Model for Computer Software", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

68) Littlewood, B., "MTBF is Meaningless in Software Reliability", IEEE Transactions on Reliability, April 1975.

69) Littlewood, B., "A Reliability Model for Markov Structured Software", Proceedings of the 1975 International Conference on Reliable Software, April 21-23, 1975.

70) Littlewood, B., "A Semi-Markov Model for Software Reliability with Failure Costs", Proceedings of the Symposium on Computer Software Engineering, 1976.

71) Losq, J., Modeling and Reliability of Redundant Digital Systems, Digital Systems Laboratory, Stanford University, Technical Report No. 58, February 1975.

72) MacWilliams, W.H., "Reliability of Large Real-Time Control Software System", 1973 IEEE Symposium on Computer Software Reliability.

73) Manley, J.H. and Lipow, M., Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group, Vol. I, AFSC-TR-75-05, November 1975.

74) Manley, J.H. and Lipow, M., Findings and Recommendations of the Joint Logistics Commanders Software Reliability Work Group, Vol. II, AFSC-TR-75-05, November 1975.

75) Meeker, R.E., Jr. and Ramamoorthy, C.V., A Study in Software Reliability and Evaluation, AFOSR-TR-73-0763, February 1973.

76) Mills, H.D., "The New Math of Computer Programming", Communications of the ACM, Vol. 18, No. 1, January 1975.

77) Miyamoto, I., "Software Reliability in Online Real Time Environment", Proceedings of the 1975 International Conference on Reliable Software, April 21-23, 1975.

78) Mohanty, S.N. and Adamowicz, M., "Proposed Measures for the Evaluation of Software", Proceedings of the Symposium on Computer Software Engineering, 1976.

79) Moranda, P.B., "Prediction of Software Reliability During Debugging", Proceedings of the 1975 Annual Reliability and Maintainability Symposium.

80) Moranda, P.B., "A Comparison of Software Error-Rate Models", McDonnell-Douglas Astronautics Company, Huntington Beach, California, November 1975.

81) Moranda, P.B., Quantitative Methods for Software Reliability Measurements, AFOSR-TR-77-0046, December 1976.

82) Moranda, P.B. and Jelinski, Z., "Software Reliability Predictions", McDonnell-Douglas Astronautics Company, Huntington Beach, California, August 1975.

83) Musa, J.D. and Hamilton, P.A., "Program for Software Reliability and System Test Schedule Estimation-Program Documentation", Bell Laboratories, Whippany, New Jersey, 1977.

84) Musa, J.D., "Program for Software Reliability and System Test Schedule Estimation - User's Guide", Bell Laboratories, Whippany, New Jersey, 1977.

85) Musa, J.D. "A Theory of Software Reliability and its Application", IEEE Transactions on Software Engineering, September 1975.

86) Myers, G.J., "Reliability Models", in Software Reliability-Principles and Practices, John Wiley and Sons, 1976.

87) Nelson, E.C., "Software Reliability", 1975 International Symposium on Fault-Tolerant Computing, June 18-20, 1975.

88) Ogdin, J.L., "Designing Reliable Software", Datamation, July 1972.

89) Paige, M.R. and Holthouse, M.A., "On Sizing Software Testing for Structured Programs", The Seventh Annual International Conference on Fault-Tolerant Computing, June 28-30, 1977.

90) Parnas, D.L., "The Influence of Software Structure on Reliability", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

91) Peterson, R.J., "Tester/1: An Abstract Model for the Automatic Synthesis of Program Test Case Specifications", Proceedings of the Symposium on Computer Software Engineering, 1976.

92) Pikul, R.A. and Wojcik, R.T., "Software Effectiveness: A Reliability Growth Approach", Proceedings of the Symposium on Computer Software Engineering, 1976.

93) Pimont, S. and Rault, J.C., "A Software Reliability Assessment Based on a Structural and Behavioral Analysis of Programs", Second International Conference on Software Engineering, October 13-15, 1976.

94) Ramamoorthy, C.V., Ho, G.S. and Han, Y.W., "Fault Tree Analysis of Computer Systems", National Computer Conference, 1977.

95) Ramamoorthy, C.V. and Ho, S.B, "Testing Large Software with Automated Software Evaluation Systems", IEEE Transactions on Software Engineering, March 1975.

96) Ramamoorthy, C.V., Meeker, R.E. and Turner, J., "Design and Construction of an Automated Software Evaluation System", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

97) Randell, B., "System Structure for Software Fault Tolerance", IEEE Transactions on Software Engineering, June 1975.

98) Rao, T.R.N. and Reinheimer, H.J., "Fault-Tolerant Modularized Arithmetic Logic Units", National Computer Conference, 1977.

99) Reifer, D.J., "A New Assurance Technology for Computer Software", Proceedings of the 1976 Annual Reliability and Maintainability Symposium.

100) Reifer, D.J. and Ettenger R.L., "Test Tools: Are They a Cure-All?" Proceedings of the 1975 Annual Reliability and Maintainability Symposium.

101) Reifer, D.J., "Automated Aids for Reliable Software", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

102) Robinson, L., Levitt, K.N., Newmann, P.G. and Saxena, A.R., "On Attaining Reliable Software for a Secure Operating System", 1975 International Conference on Reliable Software, April 21-23, 1975.

103) Rubey, R.J. Dana, J.A. and Biche, P.W., "Quantitative Aspects of Software Validation", IEEE Transactions on Software Engineering, June 1975.

104) Rubey, R.J., "Planning for Software Reliability", Proceedings of the 1975 Annual Reliability and Maintainability Symposium.

105) Rye, P., Bamberger, F., Ostanek, W., Brodeur, N. and Goode, J., Software Systems Development: A CSDL Project History, RADC-TR-77-213, June 1977.

106) Schick, G.J. and Wolverton, R.W., "Achieving Reliability in Large Scale Software Systems", Proceedings of the 1974 Annual Reliability and Maintainability Symposium.

107) Schick, G.J. and Wolverton, R.W., "An Analysis of Competing Software Reliability Models", IEEE Transactions on Software Engineering, March 1978.

108) Schneidewind, N.F., "A Methodology for Software Reliability Prediction and Quality Control", Naval Postgraduate School, Monterey, California, November 1972.

109) Schneidewind, N.F., "Analysis of Error Processes in Computer Software", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

110) Shelley, M., Computer Software Reliability - Fact or Myth?, TR-MMER/RM-73-125, Ogden Air Logistics Center, Hill AFB, Utah, November 1973.

111) Shooman, M.L., Probabilistic Reliability: An Engineering Approach, McGraw-Hill Book Company, 1968.

112) Shooman, M.L., "Structured Models for Software Reliability Prediction", Second International Conference on Software Engineering, October 13-15, 1976.

113) Shooman, M.L. and Natarajan, S., "Effect of Manpower Deployment and Bug Generation on Software Error Models", Proceedings of the Symposium on Computer Software Engineering, 1976.

114) Shooman, M.L., "Probabilistic Models for Software Reliability Prediction", Proceedings of the 1972 Symposium on Fault-Tolerant Computing.

115) Shooman, M.L. and Bolsky, M.I., "Types, Distribution, and Test and Correction Times for Programming Errors", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

116) Shooman, M.L., "Software Reliability: Measurement and Models", Proceedings of the 1975 Annual Reliability and Maintainability Symposium.

117) Shooman, M.L., "Operational Testing and Software Reliability Estimation During Program Development", IEEE Symposium on Computer Software Reliability, April 30-May 2, 1973.

118) Shooman, M.L., "Probabilistic Models for Software Reliability Prediction", in Probability Models for Software, W. Freiberger, Ed., Academic Press, 1972.

119) Shooman, M.L. and Ruston, H., Summary of Technical Progress Software Modeling Studies, RADC-TR-75-245, September 1975.

120) Shooman, M.L. and Ruston, H., Summary of Technical Progress Software Modeling Studies, RADC-TR-75-246, September 1975.

121) Shooman, M.L., "Software Reliability: Analysis and Prediction", Source unknown, 1977.

122) Siewiorek, D.P., "Reliability Modeling of Compensating Module Failures in Majority Voted Redundancy", IEEE Transactions on Computers, May 1975.

123) Sukert, A.N., "An Investigation of Software Reliability Models", Proceedings of the 1977 Annual Reliability and Maintainability Symposium.

124) Sukert, A.N., A Software Reliability Modeling Study, RADC-TR-76-247, August 1976.

125) Szabo, S.G., "A Schema for Producing Reliable Software", 1975 International Symposium on Fault-Tolerant Computing, June 18-20, 1975.

126) Tasar, O. and Tasar, V., "A Study of Intermittent Faults in Digital Computers", National Computer Conference, 1977.

127) Thayer, R.H. and Hinton, E.S., "Software Reliability - A Method That Works", National Computer Conference, 1975.

128) Thayer, T.A., Craig, G.R. and Frey, L.E., Software Reliability Study, RADC-TR-76-238, August, 1976.

129) Thayer, T.A., "Understanding Software through Empirical Reliability Analysis", National Computer Conference, 1975.

130) Trivedi, A.K. and Shooman, M.L., "A Many-State Markov Model for the Estimation and Prediction of Computer Software Performance Parameters", Proceedings of the 1975 International Conference on Reliable Software, April 21-23, 1975.

131) Trivedi, A.K. "Computer Software Reliability: Many-State Markov Modeling Techniques", Ph.D. Dissertation, Department of Electrical Engineering, Polytechnic Institute of New York, June 1975.

132) Trivedi, A.K. and Shooman, M.L., Computer Software Reliability: Many-State Markov Modeling Techniques, RADC-TR-75-169, July 1975.

133) Von Henke, F.W. and Luckham, D.C., "A Methodology for Verifying Programs", Proceedings of the 1975 Conference on Reliable Software, April 21-23, 1975.

134) Wagoner, W.L., The Final Report on a Software Reliability Measurement Study, The Aerospace Corporation, El Segundo, California, Report No. TOR-0074(4112)-1, August 15, 1973.

135) Wall, J.K. and Ferguson, P.A. "Pragmatic Software Reliability Prediction", Proceedings of the 1977 Annual Reliability and Maintainability Symposium.

136) Weiss, H.K., "Estimation of Reliability Growth in a Computer System With a Poisson-Type Failure", Operations Research, Vol. 4, No. 5, October 1956.

137) Willman, H.E., James, T.A., Beaureguard, A.A. and Hilcoff, P., Software Systems Reliability: A Raytheon Project History, RADC-TR-77-188, June 1977.

138) Wolverton, R.W. and Schick, G.J., "Assessment of Software Reliability", 11th Annual Meeting of the German Operations Research Society, September 1972.

139) Wulf, W.A., "Reliable Hardware/Software Architecture", IEEE Transactions on Software Engineering, June 1975.

140) Yau, S.S., Cheung, R.C. and Cochrane, D.C., "An Approach to Error-Resistant Software Design", Second International Conference on Software Engineering, October 13-15, 1976.

☆U.S.Government Printing Office: 1979 — 657-002/330